Apple® II Desktop Toolkit–ProDOS®

Apple® II Desktop Toolkit–ProDOS®

Apple II Desktop Toolkit

Tools and Compilers Group, Apple Computer, Inc.

consisting of:

- Developer's Handbook for the Apple II MouseText Tool Kit
- Mouse Graphics Tool Kit
- Apple II Graphics Primitives Handbook

this manual is used with the following APDA products:

ProDOS Version 1.0B5 (APDA# K2SDTD)
Pascal Version 1.0B5 (APDA# K2SDTP)

This document is in manuscript form. It does not include

- *final technical corrections*
- *final editorial corrections*
- *final art work*
- *an index.*

Copyright © 1985 Apple Computer, Inc. All rights reserved.

DEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTES
DEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTES
DEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTES

Developer's Handbook for the Apple II MouseText Tool Kit

DEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTES
DEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTES
DEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTESDEVELOPERNOTES

030-0891-2

NOTICE

Apple Computer, Inc. reserves the right to make improvements in the product described in this manual at any time and without notice.

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL OR WITH RESPECT TO THE SOFTWARE DESCRIBED IN THIS MANUAL, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. APPLE COMPUTER, INC. SOFTWARE IS SOLD OR LICENSED "AS IS". THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAMS PROVE DEFECTIVE FOLLOWING THEIR PURCHASE, THE BUYER (AND NOT APPLE COMPUTER, INC., ITS DISTRIBUTOR OR ITS RETAILER) ASSUMES THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL APPLE COMPUTER, INC. BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE, EVEN IF APPLE COMPUTER, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

This manual is copyrighted. All rights are reserved. This document may not, in whole or part, be copied, photocopied, reproduced, translated or reduced to any electronic medium or machine readable form without prior consent, in writing, from Apple Computer, Inc.

© 1985 APPLE COMPUTER, INC.
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010

The word APPLE and the Apple logo are registered trademarks of APPLE COMPUTER, INC.

Contents

Table of Contents

6 List of Figures and Tables

7 Foreword

9 Chapter 1. Introduction: The MouseText Tool Kit

10 Features Supported by the Tool Kit
10 The Cursor
10 Events
11 Menus
14 Windows
14 Parts of a Window
17 Window Coordinates
19 Document Information
21 Control Regions: the Scroll Bar
22 Interrupts and the Tool Kit
22 Lists of Tool Kit Commands
28 Mouse Emulation
28 Keyboard Mouse Mode
30 Safety-Net Mode

31 Chapter 2. Specifications of the Commands

32 Startup Commands
33 StartDeskTop
35 StopDeskTop
36 PascIntAdr
37 SetUserHook
39 Version
40 KeyBoardMouse

41 Cursor Commands
41 SetCursor
42 ShowCursor
43 HideCursor
44 ObscureCursor
45 Event-Handling Commands
45 CheckEvents
47 GetEvent
49 PostEvent
50 FlushEvents
51 SetKeyEvent
52 PeekEvent
53 Menu Commands
53 Keys in Menus
55 InitMenu
56 SetMenu
60 MenuSelect
62 MenuKey
64 HiLiteMenu
65 DisableMenu
66 DisableItem
67 CheckItem
68 SetMark
69 Window Commands
70 InitWindowMgr
72 OpenWindow
77 CloseWindow
78 CloseAll
79 GetWinPtr
80 FindWindow
81 FrontWindow
82 SelectWindow
83 TrackGoAway
84 DragWindow
86 GrowWindow
88 WindowToScreen
89 ScreenToWindow
90 WinChar
91 WinString
92 WinText
93 WinBlock
94 WinOp
95 Control Region Commands
95 FindControl
97 SetCtlMax
98 TrackThumb
100 UpDateThumb
101 ActivateCtl

103 Chapter 3. The Machine Language Interface

103 Installing the Machine Language Tool Kit
104 Syntax of Machine Language Calls
105 The Machine Language Commands
105 Startup Commands
106 Cursor Commands
107 Event-Handling Commands
108 Menu Commands
110 Window Commands
114 Control Region Commands

117 Chapter 4. The Pascal Interface

117 Installing the Pascal Interface
117 Data Structures
117 Constants and Type Definitions
126 Command Functions and Procedures
126 Startup Commands
127 Cursor Commands
128 Event-Handling Commands
129 Menu Commands
131 Window Commands
134 Control Region Commands
135 Utility Functions

137 Chapter 5. The Applesoft Interface

137 Installing the Applesoft Interface
138 Using the Ampersand Commands
139 The Ampersand Commands
139 Startup Commands
140 Cursor Commands
141 Event-Handling Commands
142 Menu Commands
145 Window Commands
151 Control Region Commands
153 Utility Commands

155 Appendix A. The AppleMouse II Interface Card

155 Passive Versus Active Operation
156 Mouse Interrupts
156 The TimeData Firmware Call

157 **Appendix B. The Mouse Firmware Interface**

157 Finding the Mouse Card
156 Reading Mouse Data
160 Operating Modes
161 Passive Mode
162 Interrupt Mode
162 Unclaimed Interrupts
163 Making Calls to Mouse Firmware
164 Parameter Passing
165 The Firmware Routines
165 SetMouse
165 ServeMouse
166 ReadMouse
166 ClearMouse
166 PosMouse
166 ClampMouse
167 HomeMouse
167 InitMouse

169 **Appendix C. The Mouse Pascal Attach Driver**

169 Installing the Mouse Pascal Attach Driver
170 About Pascal Attach Drivers
171 The Pascal Interface
173 Interrupts

175 **Appendix D. Sample Program**

175 Pseudocode Listing

179 **Appendix E. MouseText Characters**

181 **Appendix F. Tool Kit Error Codes**

List of Figures and Tables

13 Figure 1-1. Menu Components
15 Figure 1-2. Typical Display With Windows
16 Figure 1-3. Parts of a Window
17 Figure 1-4. Window With Inactive Scroll Bars
20 Figure 1-5. Location Parameters in a Document
23 Table 1-1a. Alphabetical List of Tool Kit Commands
24 Table 1-1b. Alphabetical List of Tool Kit Commands, Continued
25 Table 1-2a. Startup Commands
25 Table 1-2b. Cursor Commands
25 Table 1-2c. Event-Handling Commands
26 Table 1-2d. Menu Commands
27 Table 1-2e. Window Commands
27 Table 1-2f. Control Region Commands

54 Table 2-1. Control Keys for Menu Items
56 Table 2-2. Data Structure for a Menu Bar
57 Table 2-3. Contents of Option Byte in Each Menu Block
58 Table 2-4. Data Structure for a Menu
59 Table 2-5. Contents of Option Byte in Menu Data Structure
73 Table 2-6. Information Structure for a Window
74 Table 2-7. Contents of Window Option Byte in Window Information Structure
75 Table 2-8. Contents of Horizontal or Vertical Control Option Byte in Window Information Structure
75 Table 2-9. Contents of Window Status Byte for Window Information Structure
76 Table 2-10. Information Structure for a Documents

105 Table 3-1. Processor Status After Return From Tool Kit

159 Table B-1. Screen Locations for Mouse Data
160 Table B-2. Button and Interrupt Status Byte
161 Table B-3. Bits in the Mode Byte
164 Table B-4. Entry Point Address Bytes

170 Table C-1. Attach Files
171 Table C-2. Pascal I/O Calls

180 Figure E-1. The MouseText Icon Characters

182 Table F-1. MouseText Tool Kit Error Codes

Foreword

This is the developer's handbook for Version 2.1 of the MouseText Tool Kit for the Apple II. The main purpose of the handbook is to tell you how to use the Tool Kit routines in your application programs. In addition, the handbook includes appendixes that contain information about the mouse itself and about the hardware and software that make it work.

Version 2.1 of the Apple II MouseText Tool Kit provides support for mouse-operated menus and windows using the text display. It uses the Mouse Text icon characters available on the Apple IIc. The icon characters are available on the Apple IIe only with an updated character ROM. Another tool kit, The Mouse Graphics Tool Kit, will support the double high-resolution graphics displays on the Apple IIc and the Apple IIe.

Note to Users: This is not the owner's manual for AppleMouse II. That manual, The AppleMouse II User's Manual, tells how to install the mouse on the Apple II and describes the demonstration program that comes with the mouse. This handbook tells you how to use the MouseText Tool Kit routines in programs that you write yourself.

Chapter 1 outlines the features of the MouseText Tool Kit and tells you what the Tool Kit routines will do for your application programs.

Chapter 2 gives complete specifications for the MouseText Tool Kit commands.

Chapters 3, 4, and 5 describe how to use the MouseText Tool Kit with application programs written in each of three different languages: Chapter 3 describes the command calls in machine language, Chapter 4 describes the command procedures in Pascal, and Chapter 5 describes the

ampersand commands used in Applesoft.

The appendixes provide additional information about using the AppleMouse II. Appendix A describes the interface card that supports the operation of the mouse hardware. Appendix B describes the interface to the mouse firmware (the level of communication and control between the hardware and the Tool Kit routines). Appendix C tells you how to install the Pascal Attach Driver that adds mouse communications to the Pascal BIOS. Appendix D contains programming examples using the MouseText Tool Kit. Appendix E describes the special Mouse Text characters. Appendix F is a combined list of the error codes returned by the Tool Kit commands.

Chapter 1

Introduction: The MouseText Tool Kit

The Apple II MouseText Tool Kit is a set of software routines that you can use to implement mouse-controlled menus and text windows for your application programs. This version of the Tool Kit provides commands for displaying and controlling pull-down menus, including

- cursor selection and display
- menu bar displays
- menu item selection.

This version of the MouseText Tool Kit also has window-handling commands used in desk-top displays for handling folders and the like. These commands perform functions such as

- window selection and display
- window dragging and size changing
- writing text in windows.

The Tool Kit also provides support for programs to perform functions like scrolling windows through documents.

Special Characters: The character generator in the Apple IIc includes special characters called MouseText that can be used for cursor displays. A new character-generator ROM will be available to provide the MouseText characters on the Apple IIe. The MouseText characters are described in Appendix E.

The Tool Kit supports 80-column displays on the Apple IIc and, via the Apple 80-column text card or equivalent cards, on the Apple IIe.

Features Supported by the Tool Kit

The commands in the MouseText Tool Kit enable your program to support several kinds of features:

- The Cursor
- Events
- Menus
- Windows
- Control Regions consisting of Scroll Bars with Thumbs

The sections that follow outline these features and mention some of the individual commands your program calls. Tables 1-1 and 1-2 list all of the commands; they are described individually in Chapter 2.

The Cursor

The cursor is the character that moves on the display as the user moves the mouse. Cursor commands enable the program to select the character displayed as the cursor and to turn the cursor on and off. The Tool Kit uses the mouse to control the position of the cursor. There is also a means of controling the cursor and the Tool Kit functions by pressing keys: see the section "Mouse Emulation" later in this chapter.

Events

The Tool Kit deals with four kinds of events: mouse events, keyboard events, update events, and application events (optional with the application program). Mouse events are button pressed (down), button released (up), and moving the mouse with the button held down (drag). Mouse motion with the button up is not an event, but the program can obtain the most recent mouse position even if no event has occurred. Keyboard events are keypresses and are optional; that is, the program specifies that it handles keypresses itself or that the Tool Kit deals with them.

Update events are a special case, provided for those applications with windows that can't be refreshed automatically. Please see the description in Chapter 2 at the GetEvent command.

Precedence of Events: If the mouse button is down, the Tool Kit ignores keypresses.

The Tool Kit's event-handling commands maintain a queue of events.

The program detects mouse events by calling GetEvent. With the Tool Kit running in Passive Mode, GetEvent automatically issues an internal call to CheckEvents. The CheckEvents command posts mouse events and keypress events in the queue and updates the mouse position. If the event queue is empty, the GetEvent command simply returns the most recent mouse position.

In Interrupt Mode, the Tool Kit's interrrupt handler calls CheckEvents 60 times per second, synchronized with the display vertical blanking (VBL). In Passive Mode, the application program must call CheckEvents or GetEvent often enough to obtain smooth cursor motion. Also, the application program can put its own events into the queue by calling the PostEvent command.

CheckEvents is the only command that reads the mouse; if it is never called, either directly, indirectly by GetEvent, or (in Interrupt Mode) by the Tool Kit itself,the cursor will never move.

If the event queue fills up, the Tool Kit ignores new events until there is room for them in the queue. To empty the queue, the program calls the FlushEvents command.

Note: Frequent calls to CheckEvents also provide a type-ahead feature by posting keyboard events in the queue until the program can process them.

Menus

The Tool Kit's menu management commands enable programs to provide pull-down menus. The visible components of a menu are

- a menu bar at the top of the display, showing the menu titles
- the menu items that appear, one to a line, when a menu pops down.

When the user moves the cursor onto a title in the menu bar and presses the button on the Mouse, the application program calls the

MenuSelect command, which displays a menu and tracks the mouse as long as the mouse button stays down. The user doesn't literally pull it down: instead, it pops down as soon as the application program determines that the cursor has moved onto the title. As the user keeps the button pressed and moves the cursor down the menu, the Tool Kit highlights the item the cursor is pointing to by displaying it in inverse video.

When the user releases the button, the item that the cursor was pointing to is selected and the menu disappears. To tell the user that something is happening, the Tool Kit leaves the menu title in the menu bar highlighted. The application program turns off the highlighting of the title as soon as it finishes performing the selected operation.

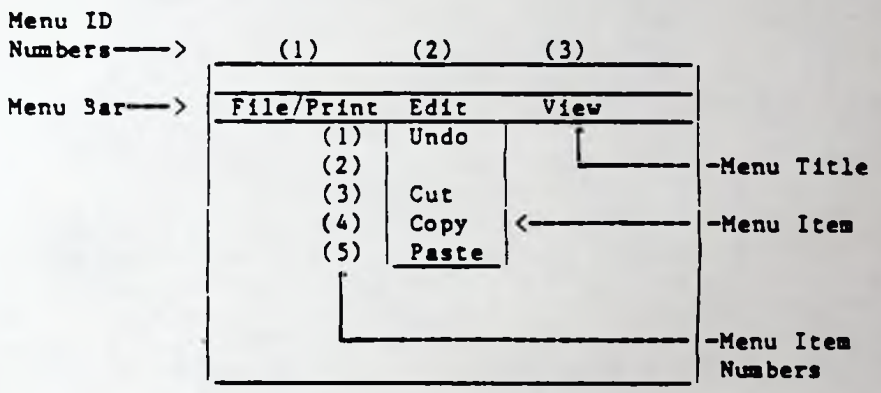
The data structures the Tool Kit uses to manage the menu information also contain information that is not displayed, but is returned to the program when a menu item is selected:

- a menu ID number for each menu

- a menu item number for each item

The program can set individual items or an entire menu to the disabled state. Disabled items or menus are not highlighted when the cursor moves onto them, and they cannot be selected.

Figure 1-1 Menu Components. Note: Numbers in parentheses are menu ID and menu item numbers and do not appear in the display. The menu item numbers are always sequential starting with 1, but the menu ID numbers can be in any order, as long as they're between 1 and 255.



The application program calls the SetMenu command with data structures containing the menu information the Tool Kit needs, and the Tool Kit displays the menus. The program can call SetMenu during the course of operation to change the contents of menus. The menu data structures are described in the "Menu Commands" section in Chapter 2.

When the FindWindow command detects the mouse button pressed in row 0 (the menu bar), the program calls the MenuSelect command. MenuSelect takes care of the entire selection process: it displays the menus and tracks the mouse position with the cursor for as long as the user holds down the mouse button. If the user selects a menu item, the MenuSelect command highlights the menu's title in the menu bar and returns the menu item number and the menu ID number. If the user doesn't select a menu item, the MenuSelect command returns a menu ID value of 0.

Keeping the selected menu title highlighted while the operation is being performed gives useful feedback to the user. After the program has carried out the selected operation, it should call HiLiteMenu with menu ID set to 0 to un-highlight the menu title.

For menu items that are used often, the program can provide fast item selection; it does this by allowing the user to press keys instead of moving the mouse. To do this, the program specifies the keys the user can press to select the items in the menus. When the GetEvent command

returns a keypress, the program calls the MenuKey command. MenuKey gets the menu ID and the item number by searching the menu data structure for a matching key, and then highlights the selected menu title the same way MenuSelect does. After the operation has been performed, the program must use the HiLiteMenu command to turn off the highlighting of the selected title.

Windows

The Tool Kit's window commands make it possible for programs to use the mouse to control multiple windows. Figure 1-2 shows how windows appear on the display screen.

Parts of a Window

Each window has several parts, as shown in Figure 1-3. The two main parts of a window are the drag bar at the top, including the title of the window, and the content region, where the application displays information. The drag bar is used for moving the window around on the display. To move the window, the user positions the cursor on the drag bar and holds down the mouse button while moving the cursor to the desired position. The drag bar also contains the Close Box, or Go-Away Box. To close the window, the user clicks and releases the mouse while in the Go-Away Box.

The lower-right corner of the window contains the Grow Box, which is used to change the size of a window. To do this, the user presses the mouse button when the cursor is in the Grow Box, then holds the button down while moving the mouse. The display shows the new size of the window as an outline that moves around as the mouse moves. When the user releases the mouse button, the Tool Kit redisplays the window with its new size but without contents. The program puts appropriate text into the re-sized window by calling window commands or its own window subroutines.

Figure 1-2
Typical Display With Windows

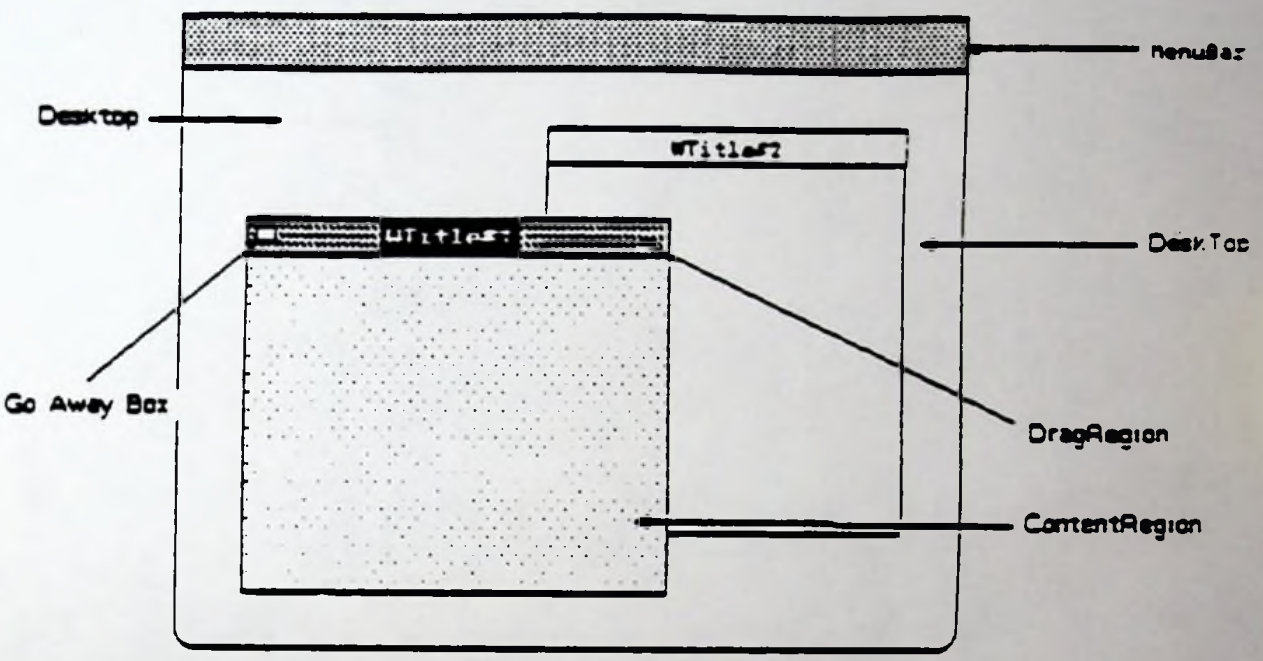


Figure 1-3 Parts of a Window

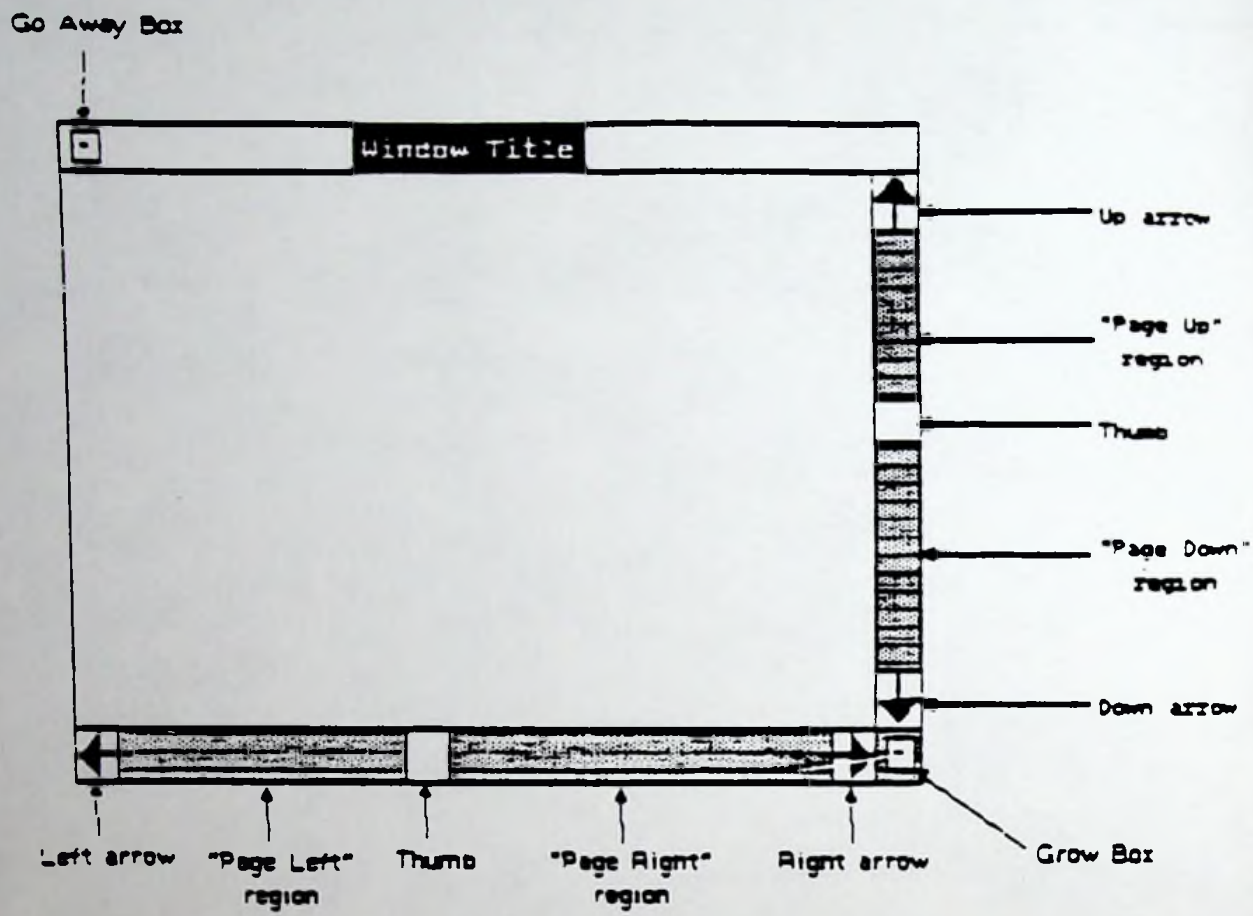
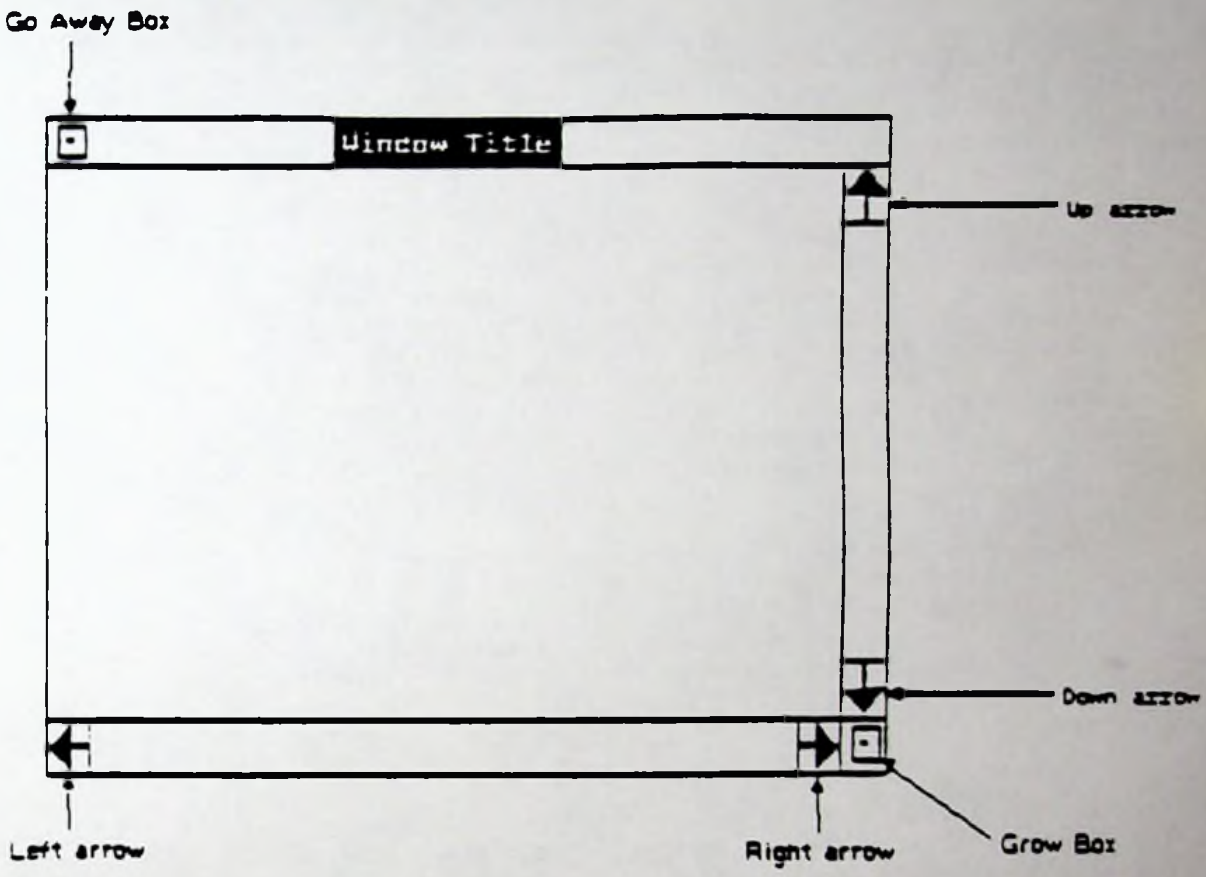


Figure 1-4
Window With Inactive Scroll Bars



Window Coordinates

Three different coordinate systems are used with the window commands:

- mouse coordinates, with X from 0 to 79 and Y from 0 to 23
- screen coordinates, with X from -80 to 159 and Y from -24 to 47
- window coordinates, with X from -80 to 159 and Y from -24 to 47

The mouse coordinates correspond to the absolute range of the display screen and are expressed as unsigned byte quantities. The window and screen coordinates are represented as two-byte signed quantities.

It is important to be aware of the ranges of the signed two-byte quantities because the Tool Kit routines make certain assumptions about the high byte. The only time the high byte is not simply the sign extension of the low byte's sign bit is when the value is in the range 128 to 159 for the X axis. The Y-axis quantities are also two-byte quantities for the sake of consistency. The only legal values of the high byte are $ØØ and $FF.

To be visible, characters must be in the top window, and their screen coordinates must be in the range from Ø to 79 in the X axis and Ø to 23 in the Y axis. What's more, if the width of the window is W and the length of the window is L, characters are visible only if their window coordinates are in the range from Ø to W - 1 in the X axis and Ø to L - 1 in the Y axis. The scroll bars are considered to be in the content area, so the useful content area range is from Ø to W - 3 if the vertical scroll bar space is used. Similarly, if there is a horizontal scroll bar, the useful content area range is from Ø to L - 2 in the Y axis.

Note: If a Grow Box is present, the vertical scroll bar space is used even if the scroll bar is not present. This ensures that the useful content area is always rectangular.

There must be at least one character in the window's content area for a Window Information Data Structure to be displayed correctly. The window length must be at least 1, or 2 if there is a horizontal scroll bar. Window width must be at least one, or three if there is a vertical scroll bar or a Grow Box. The maximum window width is 8Ø. The maximum length is 22 for normal windows, 23 for dialog windows.

Note: It is a good idea to keep window width greater than 3, else you can have a window whose title does not show, or even a window that cannot be dragged, but only closed, because there is only space enough for the Close Box.

A window can be placed in any position on the screen, including positions that make part of the window invisible. This is the reason for the ranges of the screen and window coordinates. Even though the ranges normally used are positive, you can get meaningful negative

values when you convert from one coordinate system to another. For example, a window's drag bar is always in the negative range of the window's Y axis.

Note: Windows are output-only devices; the Tool Kit will not copy their contents into user memory. The application program must ensure that the information in the content memory area and the contents of the window agree.

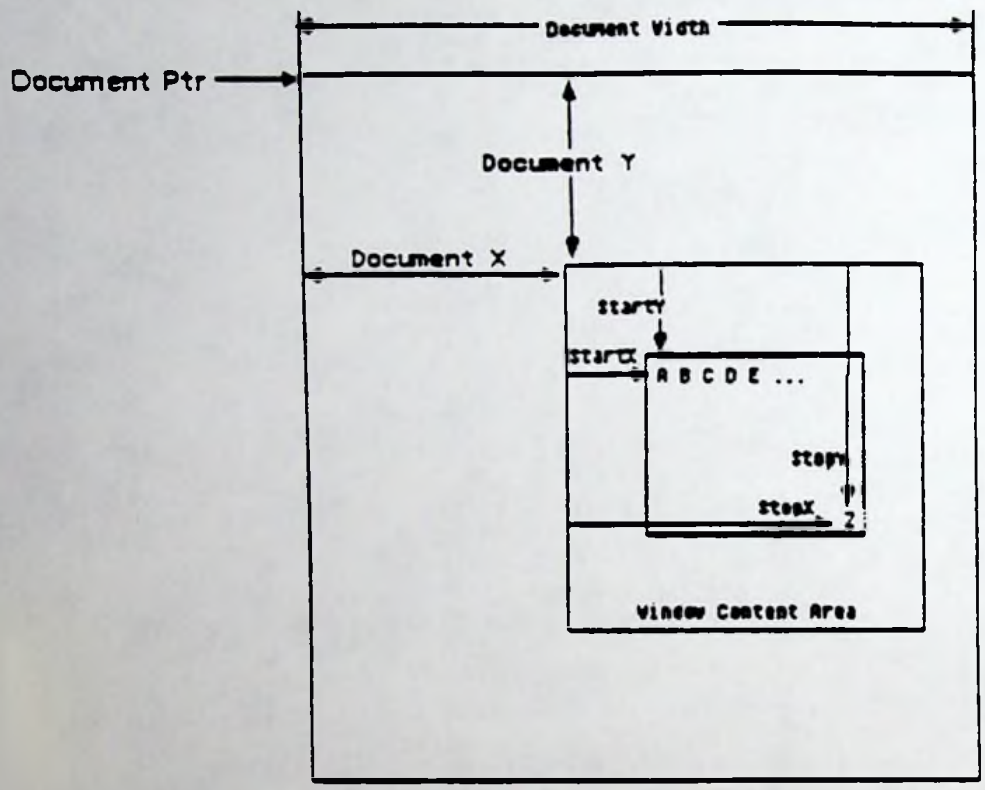
Document Information

The only document display feature built into the Tool Kit is a screen image of the text. Each line is padded with spaces on the right, and there are no special line delimiters; the number of characters per line is fixed.

To support the document display, the window management part of the Tool Kit needs certain information about the document. This information is in the Document Information Data Structure (Dinfo), described in Chapter 2. The location of the window in the document is specified by Dinfo quantities Dx and Dy (see Figure 1-5). The window can be placed anywhere within the document. In this sense, the document dimensions can be considered as a fourth coordinate system in which the window coordinates are embedded.

Other kinds of document displays are possible, but the routines to create them must be provided by the application program. For information about adding display routines, see the "SetUserHook" section in Chapter 2.

Figure 1-5
Location Parameters in a Document



Whenever a window is dragged, the Tool Kit must redisplay the content areas of the windows. The application program can override the Tool Kit's document display feature by having a routine that is called by the Tool Kit whenever the window is to be redisplayed. The program passes the address of this assembly language routine to the Tool Kit as part of the Window Information Data Structure, described in Chapter 2. Because of the way the Tool Kit saves zero-page locations, the program's routine cannot rely on the contents of those zero-page locations. Furthermore, the routine can only call the Tool Kit's window update commands to update the content region. These commands are WinChar, WinString, WinText, WinBlock, and WinOp. (Note: WinBlock uses a Document Information Data Structure.)

In the case where the window should not be refreshed automatically, the Tool Kit uses a type of event called an update event to signal the application that the window needs to be refreshed. The application specifies that a window is of this type by making the two-byte DInfo pointer (in the Window Data Structure) equal to zero. Please see the description in Chapter 2, under the GetEvent command.

Control Regions: The Scroll Bar

The only window control regions supported by the Tool Kit are the scroll bars displayed in the content region of the front (active) window (see Figure 1-3). Either horizontal or vertical scroll bars or both may be present.

An active scroll bar has several components, as shown in Figure 1-3. There are arrows at either end, an open box called the Thumb, and gray regions in between called Page-Up and Page-Down Regions in a vertical scroll bar, and Page-Left and Page-Right Regions in a horizontal scroll bar. (Sometimes the gray regions are called Page-Up and Page-Down Regions even in a horizontal scroll bar.)

An application program should support three different ways of scrolling the window contents using the scroll bars:

- Pressing the mouse button with the mouse in the arrows to scroll continuously as long as the button is down. The thumb moves to indicate the relative position of the window in the document.

- Pressing the mouse button in the Thumb itself, and dragging the Thumb to cause the corresponding scrolling of the document.

- Pressing the mouse button in a Page-Up or Page-Down Region to scroll the window up or down a page—that is, a window full—at a time.

The Thumb should appear right next to one of the arrows only when the first or last character of the document appears in the window. This ensures that the user can always page and that the Thumb can be used to get to the first and last characters of a document.

If the full width or length of a document appears in the window, the program should make the scroll bars reflect this by putting them into the inactive state, which shows the arrows, but no page regions or Thumb (see Figure 1-4).

If the window is so narrow that fewer than three character cells are available for the page regions and the Thumb, the Tool Kit will not display the Thumb. If fewer than three cells are available for the entire scroll bar, not even the arrows will show, and the user will be unable to scroll. Instead, the Tool Kit will display a gray region if the scroll bar is active, or a hollow region if it is inactive.

Interrupts and the Tool Kit

Follow this sequence of steps to start the mouse:

(1) (For Pascal only) Call PascIntAdr to get the address of the Tool Kit's interrupt handler.

(2) (For Pascal only) Pass the interrupt address to the mouse firmware by calling SetMouse as described in Appendix B, "The Mouse Firmware Interface." Mouse Mode should be set to passive.

(3) Call StartDesktop with the UseInterrupts parameter set the way you want it for your program.

(4) (optional) Call SetUserHook to pass the addresses of your program's interrupt handlers, if any, to the Tool Kit.

The Tool Kit saves the interrupt state of the machine when your program calls the StartDeskTop command. When the program calls the StopDeskTop command, the Tool Kit sets the state of the machine to the state saved by StartDeskTop.

When you use the Tool Kit in Interrupt Mode, the Tool Kit provides the interrupt handler. In addition, the Tool Kit allows the application program to have interrupt handler subroutines that are called by the Tool Kit. The program passes each subroutine's address to the Tool Kit as a parameter by calling the SetUserHook command. This feature makes it possible for the application program to perform tasks at interrupt time.

A user hook routine that is called at interrupt time cannot call most Tool Kit commands. Doing so could put the Tool Kit into an unknown state. If a program needs to generate calls to the Tool Kit because of an interrupt, the interrupt routine should set a flag that the program checks during its main polling loop.

Lists of Tool Kit Commands

Tables 1-1 and 1-2 list all of the Tool Kit commands by name and function. For complete descriptions of the commands, please see Chapter 2. For the commands as called in the different languages, please see the chapter devoted to the appropriate language interface.

Table 1-1a.
Alphabetical List of Tool Kit Commands

Name	Number	Type
ActivateCtl	43	Control Region Commands
CheckEvents	5	Event-Handling Commands
CheckItem	16	Menu Commands
CloseAll	25	Window Commands
CloseWindow	24	Window Commands
DisableItem	15	Menu Commands
DisableMenu	14	Menu Commands
DragWindow	30	Window Commands
FindControl	39	Control Region Commands
FindWindow	26	Window Commands
FlushEvents	07	Event-Handling Commands
FrontWindow	27	Window Commands
GetEvent	6	Event-Handling Commands
GetWinPtr	45	Window Commands
GrowWindow	31	Window Commands
HideCursor	4	Cursor Commands
HiLiteMenu	13	Menu Commands
InitMenu	9	Menu Commands
InitWindowMgr	22	Window Commands
KeyboardMouse	48	Startup Commands
MenuKey	12	Menu Commands
MenuSelect	11	Menu Commands
ObscureCursor	44	Cursor Commands
OpenWindow	23	Window Commands
PascIntAdr	17	Startup Commands

Table 1-1b.
Alphabetical List of Tool Kit Commands,
continued

Name	Type	
PeekEvent	21	Event-Handling Commands
PostEvent	46	Event-Handling Commands
ScreenToWindow	33	Window Commands
SelectWindow	28	Window Commands
SetCtlMax	40	Control Region Commands
SetCursor	2	Cursor Commands
SetKeyEvent	8	Event-Handling Commands
SetMark	20	Menu Commands
SetMenu	10	Menu Commands
SetUserHook	47	Startup Commands
ShowCursor	3	Cursor Commands
StartDeskTop	0	Startup Commands
StopDeskTop	1	Startup Commands
TrackGoAway	29	Window Commands
TrackThumb	41	Control Region Commands
UpDateThumb	42	Control Region Commands
Version	19	Startup Commands
WinBlock	36	Window Commands
WinChar	34	Window Commands
WinOp	37	Window Commands
WindowToScreen	32	Window Commands
WinString	35	Window Commands
WinText	38	Window Commands

Table 1-2a. Startup Commands

Number	Name	Description
0	StartDeskTop	Activates the mouse and the Tool Kit routines
1	StopDeskTop	Inactivates the mouse and the Tool Kit routines
17	PascIntAdr	Gets the interrupt handler address for Pascal (not applicable to BASIC)
47	SetUserHook	Sets the address of the user's interrupt handler
19	Version	Returns the version and revision numbers of the Tool Kit
48	KeyboardMouse	Conditions Tool Kit to perform next operation in emulation mode

Table 1-2b. Cursor Commands

Number	Name	Description
2	SetCursor	Sets the character used for displaying the cursor
3	ShowCursor	Makes the cursor visible
4	HideCursor	Makes the cursor invisible
44	ObscureCursor	Makes the cursor invisible until the mouse moves

Table 1-2c. Event-Handling Commands

Number	Name	Description
5	CheckEvents	Reads the mouse, moves the cursor to the new position, and posts event, if any
6	GetEvent	Gets next event; if none, gets mouse position
46	PostEvent	Posts an event in the event queue
7	FlushEvents	Empties the event queue
8	SetKeyEvent	Specifies whether Tool Kit handles keyboard events
21	PeekEvent	Returns event data without removing it from the queue

Table 1-2d. Menu Commands

Number	Name	Description
9	InitMenu	Allocates memory for temporary screen save
10	SetMenu	Initializes a menu bar data structure and displays the menu bar
11	MenuSelect	Interacts with mouse to display menu and return selection, if any
12	MenuKey	Selects a menu item to match a keypress
13	HiLiteMenu	Turns highlighting of menu title on or off
14	DisableMenu	Inhibits highlighting and selection over a whole menu
15	DisableItem	Inhibits highlighting and selection of a menu item
16	CheckItem	Turns checkmark next to item on or off
20	SetMark	Sets the character to use as checkmark

Table 1-2e. Window Commands

Number	Name	Description
22	InitWindowMgr	Initializes the open window list and buffer area
23	OpenWindow	Passes the Tool Kit a pointer to a Window Information Data Structure
24	CloseWindow	Deletes a window
25	CloseAll	Deletes all windows
45	GetWinPtr	Gets the pointer to the Window Information Data Structure (not applicable to Pascal)
26	FindWindow	Finds the window region that contains a given point
27	FrontWindow	Returns the ID number of the front window
28	SelectWindow	Makes a window the front (active) window
29	TrackGoAway	Returns whether the mouse button was released in a Go-Away Box
30	DragWindow	Displays window outline during drag, then redisplays windows
31	GrowWindow	Displays window outline during grow, then redisplays windows
32	WindowToScreen	Converts window coordinates to screen coordinates
33	ScreenToWindow	Converts screen coordinates to window coordinates
34	WinChar	Writes a character in a window
35	WinString	Writes a string in a window
38	WinText	Writes text in a window
36	WinBlock	Writes a block of text in a window
37	WinOp	Clears all or part of a window

Table 1-2f. Control Region Commands

Number	Name	Description
39	FindControl	Returns whether the mouse is in a control region
40	SetCtlMax	Sets the range of a scroll bar
41	TrackThumb	Tracks the Thumb until the mouse button is released
42	UpdateThumb	Displays the Thumb in a given position
43	ActivateCtl	Changes the state of a scroll bar (active or inactive)

Mouse Emulation

Although the menu and window capabilities of the Apple II MouseText Tool Kit are normally used with the AppleMouse II, it is possible to run a program using the Tool Kit on a computer that doesn't have a mouse. It is also possible to use the keyboard to control the menus and windows, even on a computer that has a mouse. Even when mouse emulation is going on, the Tool Kit still responds to mouse movement and button operation.

The first method of mouse emulation is called Keyboard Mouse Mode. It enables the application to support menu selection and window manipulation by means of keyboard commands. Note that the application must include the appropriate calls to provide this feature, in contrast to the Safety-Net Mode, which is transparent to the application.

The second method of mouse emulation, Safety-Net Mode, is provided for use with a computer that doesn't have a mouse. This might happen, for example, when a dealer needs to demonstrate an application and a mouse-equipped computer is inoperative or is not available.

Keyboard Mouse Mode

The Keyboard Mouse Mode of mouse emulation makes it possible for applications to provide keyboard commands for operations that normally require the mouse: selecting menus, and dragging and changing the size of windows. The choice of commands for selecting these mouseless operations is up to the application program. The recommended key sequences for use in English-speaking countries are:

ESC to get the menu display.
OPEN-APPLE-D or SOLID-APPLE-D to drag the window.
OPEN-APPLE-G or SOLID-APPLE-G to grow the window.

When the Tool Kit is in Keyboard Mouse Mode, it is performing one of those three operations and remains in Keyboard Mouse Mode until the operation is completed. Unlike Safety-Net Mode, the user doesn't have to hold a key down.

When the user initiates Keyboard Mouse Mode, the Tool Kit makes the cursor visible, even if it was previously hidden or obscured. When the keyboard operation is completed, the Tool Kit returns the cursor to its previous state of visibility.

In Keyboard Mouse Mode, the cursor keys move the cursor around on the display. If the user is doing a drag or grow, the OPEN-APPLE key acts as an accelerator for the cursor keys. With the OPEN-APPLE key down, pressing left or right arrow keys moves the cursor sideways by 10 spaces at a time. Likewise, the up and down arrow keys move the cursor up and down 5 rows at a time.

The user can terminate a Keyboard Mouse Mode operation in four different ways:

- by pressing the ESC key.
- by pressing the RETURN key.
- by pressing a valid command key.
- by pressing and releasing the mouse button.

When the user presses the ESC key, the Tool Kit cancels the operation and returns the cursor to its former position.

When the user presses the RETURN key, the Tool Kit completes the operation.

When the user presses a valid command key, the Tool Kit terminates the operation and then posts an event for the command key. If the operation was a menu selection, the Tool Kit cancels the operation. If the operation was a drag window or a grow window, the Tool Kit completes the operation. In any case, the Tool Kit returns the cursor to its original position.

When the user presses and releases the mouse button, the mouse button up event signals completion of the operation and initiates execution of the selected command, just as if the mouse had been used throughout.

After menu selection, the Tool Kit records the position of the cursor (that is, the item that is highlighted) and returns to that position (and item) when the user selects the menu again.

The three operations that can be performed in Keyboard Mouse Mode are selecting from the menu, dragging a window, and growing window. To perform these operations normally, the application calls the appropriate command: MenuSelect, DragWindow, or GrowWindow, respectively. To perform one of them in Keyboard Mouse Mode, the application must first call the KeyboardMouse command, then call the MenuSelect, DragWindow, or GrowWindow command. This causes the Tool Kit to perform the command in Keyboard Mouse Mode, functioning in the manner described above. The KeyboardMouse command has no parameters.

There is an alternative way for the application to get into Keyboard Mouse Mode, and that is calling the MenuKey command with ESC as the keystroke. That has the same effect as calling KeyboardMouse followed by MenuSelect: it initiates a menu select operation in Keyboard Mouse Mode.

Safety-Net Mode

Safety-Net Mode uses inputs from the keyboard to provide the usual mouse operations of moving the cursor around on the desktop and selecting menus. When the Tool Kit is in Safety-Net Mode, the application program works normally: all command calls are the same, and the program need not even take into account the fact that there is no mouse.

The user puts the Tool Kit into Safety-Net Mode by pressing the OPEN-APPLE key and holding it down, then pressing and releasing the SOLID-APPLE key. The Tool Kit generates a click to acknowledge that it is in Safety-Net Mode. The Tool Kit remains in Safety-Net Mode as long as the user continues to hold down the OPEN-APPLE key.

In Safety-Net Mode, the cursor keys take the place of the mouse for moving the cursor around. Each time you press a cursor key, the cursor moves one space in the direction indicated on the key. The cursor keys do not have wrap-around: when you have moved the cursor all the way to a screen edge, pressing the same cursor key again has no effect.

In Safety-Net Mode, the SOLID-APPLE key takes the place of the mouse button. Pressing the SOLID-APPLE key is like pressing the mouse button.

Note: All during Safety-Net Mode, the Tool Kit reads the cursor keys and the SOLID-APPLE key even if the application program has specified that the keyboard is to be ignored.

Chapter 2

Specifications of the Commands

This chapter gives the command number and the contents of the command list for each of the Tool Kit commands. Each command occupies a separate page and has the format shown here:

Name

Function:

A brief statement of the command's function

Command Number:

The number, in decimal and hex, that specifies the command

Parameter List:

A description of the parameters being passed

Description:

A full description of the command

Error Codes:

A list of error codes for the command

In addition to the error codes listed with the commands, there are three generic error codes that can be returned by any command. These error codes are

1 ($01) Illegal command number

2 ($02) Wrong number of parameters

3 ($03) StartDeskTop hasn't been called

All of the error codes are listed together in Appendix F.

Startup Commands

A program will normally call these commands once to set up the operating environment for the program. For example, calling the Version command tells the program which version of the Tool Kit it is using.

Your program calls the StartDeskTop command to activate the mouse and set the Operating Mode for the Tool Kit, and the StopDeskTop command to deactivate the mouse and the Tool Kit.

Pascal programs must also call the PascIntAdr command to get the address of the Tool Kit's interrupt handler so the Pascal interface can install the interrupt handler. (See Chapter 4, "The Pascal Interface.")

StartDeskTop

Function:

StartDeskTop initializes the mouse and the Tool Kit routines.

Command Number: 0 ($00)

Parameter List:

6 (input, byte) the number of parameters

id (input, byte) machine ID byte: $06 = Apple IIe or IIc

sid (input, byte) subsidiary ID byte:
$EA = Apple IIe
$E0 = Apple IIe with revised ROM
$00 = Apple IIc

op (input, byte) operating system byte:
0 = ProDOS
1 = Pascal

s# (input or output, byte) slot number of the mouse card

int (input or output, byte) interrupt usage:
0 = Passive Mode only
1 = use interrupts

col (input, byte) number of text columns:
0 = 40 columns
1 = 80 columns

Description:

StartDeskTop saves the current state of the computer, initializes the Tool Kit routines, and activates the mouse card. If the calling program specifies a slot number of 0, StartDeskTop will check the slots for a mouse card and use the first one it finds, returning its slot number in s#. If no mouse card is found, StartDeskTop will set Passive Mode and return the int parameter as 0.

If the program requires that the mouse card be present, it should set the high bit of the s# parameter on before calling StartDeskTop. When that bit is set, StartDeskTop will return an error condition if it doesn't find a mouse card.

If the program uses interrupts, it must set the int parameter to 1.

The ID bytes are the values found at locations $FBB3 and $FBC0 in the Apple IIe and Apple IIc. Version 2 of the MouseText Tool Kit requires the machine ID byte to be $06.

The Tool Kit doesn't do anything about the 80-column firmware. The program has to activate the firmware if it is needed.

StartDeskTop sets the cursor to the arrowhead character (ASCII value $02) and sets it hidden; after calling StartDeskTop, an application program can call ShowCursor immediately.

Error Codes:

4 ($04)	Machine or operating system not supported
5 ($05)	Invalid slot number (less than 0 or greater than 7)
6 ($06)	Card not found
11 ($0B)	Could not install interrupt handler

StopDeskTop

Function:

StopDeskTop deactivates the mouse and the Tool Kit routines.

Command Number: 1 ($Ø1)

Parameter List:

Ø (input, byte) the number of parameters

Description:

StopDeskTop hides the cursor, removes the link to the interrupt handler, and sets the mouse to an inactive state. StopDeskTop then restores the computer to the initial state that was saved by StartDeskTop.

Error Codes:

(none)

PascIntAdr

Function:

PascIntAdr returns the address of the Tool Kit's interrupt handler.

Command Number: 17 ($11)

Parameter List:

I	(input, byte) the number of parameters
Adr	(output, word) the address of the interrupt handler

Description:

PascIntAdr returns the address of the Tool Kit's interrupt handler in the Adr parameter. Your Pascal program can pass that address on to the Mouse Attach Driver when it calls SetMouse. The SetMouse call should always specify Passive Mode along with the interrupt address. The program should do this before calling StartDesktop, which will enable interrupts if its Int parameter is set to 1.

See also Chapter 4, "The Pascal Interface."

Note: This command is used only in Pascal programs.

Error Codes:

(none)

SetUserHook

Function:

SetUserHook sets the address of the user's interrupt handler.

Command Number: 47 ($2F)

Parameter List:

2 (input, byte) the number of parameters

Id (input, byte) the ID number of the interrupt routine

Adr (input, word) the address of the interrupt routine

Description:

The SetUserHook command sets the starting address of the application program's interrupt handler routine so that the Tool Kit can pass control to it whenever CheckEvent is called. In Interrupt Mode, the Tool Kit calls CheckEvent internally during interrupt servicing, so routines installed by SetUserHook become interrrupt service routines for the application.

CheckEvent can pass control to the program's interrupt routine either before or after it checks events. The Id parameter determines at which point CheckEvent will call the interrupt routine. If Id = 0, CheckEvent will call the interrupt routine before checking events, and if Id = 1, CheckEvent will call it after checking events. In this way there can be an interrupt routine either before or after event checking, or there can be two user routines, one before and one after event checking.

If the interrupt routine that is called before event checking (Id = 0) returns to the Tool Kit with the carry flag clear, CheckEvent will not check events. This allows the application program to handle event checking itself and bypass event checking by the Tool Kit.

If the Adr parameter is set to 0, SetUserHook removes any routine previously installed.

WARNING
The user interrupt routine can call only Tool Kit commands PostEvent, ShowCursor, HideCursor, and SetCursor. Calling any other commands from the user interrupt routine could put the Tool Kit into an unknown and bizarre state.

Error Codes:

21 ($15) Illegal Id parameter (must be Ø or 1)

Version

Function:

Version returns the Tool Kit's version and revision numbers.

Command Number: 19 ($13)

Parameter List:

2 (input, byte) the number of parameters

Ver (output, byte) the version number of the Tool Kit

Rev (output, byte) the revision number of the Tool Kit

Description:

The Version command returns the version and revision numbers of the Tool Kit. The program can use these numbers to determine compatibility.

Error Codes:

(none)

KeyboardMouse

Function:

KeyboardMouse makes the next command work in mouse emulation mode, if the command is one of the three that work in that mode.

Command Number: 48 ($30)

Parameter List:

0 (input, byte) the number of parameters

Description:

The KeyboardMouse command is a function call; it has no parameters.

The KeyboardMouse command is used in conjunction with the three commands that can operate in mouse emulation mode: MenuSelect, DragWindow, and GrowWindow. To make one of these three commands operate in emulation mode, all you have to do is call the KeyboardMouse command just before calling one of them.

An application can also get this form of mouse emulation on the MenuKey command by calling the command with the ESC key as the keystroke. That has the same effect as calling KeyboardMouse and then calling MenuSelect.

Error Codes:

(none)

Cursor Commands

Your program calls these commands to select the character to display as the cursor and to turn the cursor on and off.

SetCursor

Function:

SetCursor sets the character used for displaying the cursor.

Command Number: 2 ($02)

Parameter List:

l (input, byte) the number of parameters

cc (input, byte) character to use as cursor

Description:

SetCursor sets the character displayed as the cursor. Characters normally used as the cursor include the following Mouse Text characters:

Arrowhead (ASCII value 02 $02)
Hourglass (ASCII value 03 $03)
Checkmark (ASCII value 04 $04)
Text Cursor (ASCII value 20 $14)
Cell Cursor (ASCII value 29 $1D)

If the cursor is visible, it changes to the new character as soon as SetCursor is called. Each time the cursor is moved, if it is visible, the Tool Kit saves the character at the new cursor position and replaces it with the character specified by SetCursor.

Error Codes:

(none)

ShowCursor

Function:

ShowCursor makes the cursor visible.

Command Number: 3 ($Ø3)

Parameter List:

Ø (input, byte) the number of parameters

Description:

ShowCursor makes the cursor visible. If the cursor is obscured, ShowCursor has no effect.

Error Codes:

(none)

HideCursor

Function:

HideCursor makes the cursor invisible.

Command Number: 4 ($Ø4)

Parameter List:

Ø (input, byte) the number of parameters

Description:

HideCursor makes the cursor invisible. If the cursor is obscured, ShowCursor has no effect.

Error Codes:

(none)

ObscureCursor

Function:

ObscureCursor makes the cursor temporarily invisible.

Command Number: 44 ($2C)

Parameter List:

0 - (input, byte) the number of parameters

Description:

ObscureCursor makes the cursor invisible until the mouse moves; then the cursor reappears. An appropriate time to use OscureCursor is when text is being entered, to keep the cursor from obstructing the view of the text. As soon as the user moves the mouse to perform another task, the cursor reappears.

Error Codes:

(none)

Event-Handler Commands

The Tool Kit's event-handler commands maintain an event queue for mouse and keyboard events. The CheckEvents command posts events in the queue and updates the mouse position. The GetEvents command gets the next event in the queue.

CheckEvents

Function:

CheckEvents reads the mouse, moves the cursor to the new mouse position, and posts an event, if any.

Command Number: 5 ($05)

Parameter List:

0 (input, byte) number of parameters

Description:

CheckEvents reads the mouse and posts a mouse event if the button state changed. If a key on the keyboard is down and keypress events are to be checked, CheckEvents posts a keypress event and clears the keyboard strobe. (If a previous call to SetKeyEvent has disabled keypress events, CheckEvents ignores the keypress.) CheckEvents also updates the cursor position to the X and Y values from the mouse.

If the program is using Interrupt Mode, the interrupt handler calls CheckEvents. If the program calls CheckEvents in Interrupt Mode, the Tool Kit returns an error.

In Passive Mode, the GetEvent command calls CheckEvents internally. If the program is using Passive Mode, it should call CheckEvents or GetEvent often to ensure smooth cursor motion.

Remember: CheckEvents is the only command that reads the mouse and updates the cursor position.

An application program can have an interrupt-service routine of its own that augments or even replaces the functions of CheckEvents. CheckEvents can pass control to the routine either before or after event checking. The program can even have two interrupt routines, one called before event checking and one after. See the SetUserHook command in the "Startup Commands" section to see how to do this.

Error Codes:

7 ($07)	Interrupt Mode in use. (Program specified Interrupt Mode in StartDeskTop, so it can't call CheckEvents.)

GetEvent

Function:

GetEvent fetches the next event from the event queue. If there is none, GetEvent returns the mouse position. In Passive Mode, GetEvent calls CheckEvents.

Command Number: 6 ($Ø6)

Parameter List:

3 (input, byte) number of parameters

et (output, byte) event type:
Ø = no event
1 = button down
2 = button up
3 = key pressed
4 = drag event
5 = Apple key down
6 = update event

eb1 (output, byte) event byte 1: X coordinate or key value

eb2 (output, byte) event byte 2: Y coordinate or key modifier

Description:

GetEvent fetches the next event from the event queue so the program can respond to the pressing of a key or the mouse button. In Passive Mode, GetEvent calls CheckEvent internally to make sure the lastest event gets processed.

The event-type variable is a byte that indicates what happened to cause the event. If the event type is Ø, 1, 2, 4, or 5, the event bytes are the X and Y coordinates of the mouse position from the last call to CheckEvents. If the event type is 3, the event bytes are the key and the key modifier. The high bit of the key value is Ø. The key modifier values are:

Ø = no modifier
1 = OPEN-APPLE pressed
2 = SOLID-APPLE pressed
3 = both Apple keys pressed

The drag event (et parameter = 4) is similar to a no event except that the mouse button is still down. After getting a button-down event,

the program should get drag events or a button-up event. If the program gets a no event while waiting for a button-up event, that indicates that a mouse-up event was missed and that you don't know what the mouse position was at that time (you only know its present position). If this happens, the program must cancel any operation that is in progress.

The Apple-key down event indicates that one of the Apple keys was down when the mouse button was pressed.

By the Way: This is similar to the shift click event on a Lisa or a Macintosh. We can't read the shift keys on the Apple II, but we can read the Apple keys.

An event type of 6 indicates an update event. This indicates that a window that cannot be automatically refreshed needs updating. The window ID is returned in ebl, the key value parameter. This event only occurs when the application has set the DInfo pointer in the Window Data Structure to zero, indicating that the window cannot be automatically refreshed.

Error Codes:

(none)

PostEvent

Function:

PostEvent posts an event into the event queue.

Command Number: 46 ($2E)

Parameter List:

3 (input, byte) number of parameters

et (input, byte) event type:
Ø = no event
1 = button down
2 = button up
3 = key pressed
4 = drag event
5 = Apple key down
6 = update event

eb1 (input, byte) event byte 1: X coordinate or key value

eb2 (input, byte) event byte 2: Y coordinate or key modifier

Description:

PostEvent posts an event into the event queue. The parameter list is the same as for GetEvent except that all of the parameters are inputs.

PostEvent can have an event type like the ones returned by GetEvent (et = Ø, 1, ...5) or it can have a type defined by the application program (et = 128, 129, ...255). Any other value for the et parameter is illegal. The Tool Kit ignores events of type 128-255.

Error Codes:

19 ($13) The event queue is full; the event was not posted.

2Ø ($14) Illegal event type; the event was not posted.

FlushEvents

Function:

FlushEvents empties the event queue.

Command Number: 7 ($Ø7)

Parameter List:

Ø (input, byte) number of parameters

Description:

FlushEvents empties the event queue.

Error Codes:

(none)

SetKeyEvent

Function:

SetKeyEvent specifies whether Tool Kit treats keypresses as events.

Command Number: 8 ($08)

Parameter List:

l (input, byte) number of parameters

sk (input, byte) set keyevent:
Ø = don't check keyboard,
1 = check the keyboard

Description:

SetKeyEvent specifies whether Tool Kit posts keypresses as events. If the value of sk is 1, the Tool Kit reads the keyboard. If a key is pressed, the Tool Kit posts a key event and clears the key strobe. If the value of sk is Ø, the Tool Kit doesn't handle keypresses. At start up, the Tool Kit is set to post keyboard events.

The Tool Kit handles keypresses as events in the queue, providing a form of type-ahead. This means that Pascal programs don't need the Keypress function in the Applestuff Unit as long as they're using the Tool Kit.

Error Codes:

(none)

PeekEvent

Function:

PeekEvent reports on the next event without removing it from the queue.

Command Number: 21 ($15)

Parameter List:

3	(input, byte) number of parameters
et	(output, byte) event type: 0 = no event 1 = button down 2 = button up 3 = key pressed 4 = drag event 5 = Apple key down 6 = update event
eb1	(output, byte) event byte 1: X coordinate or key value
eb2	(output, byte) event byte 2: Y coordinate or key modifier

Description:

PeekEvent returns information from the next event in the event queue, but does not remove the event from the queue. The parameters are the same as for the GetEvent command, described earlier.

Error Codes:

(none)

Menu Commands

The Tool Kit's menu commands provide menu display and selection functions. Once you have set up the menu data structures with SetMenu, the MenuSelect command will display a menu, track the mouse and move the cursor, highlight menu items as the cursor moves onto them, return with the menu ID and item numbers selected, and leave the menu title highlighted. Other menu commands inhibit menus or menu items and display a checkmark beside specified menu items.

It is the responsibility of the application program to ensure that menu titles do not extend past the right edge of the screen. The program must make sure that a menu's width is always less than the screen width minus two (38 or 78) and that a menu's length is always less than screen length minus two (22). Otherwise, the menu routines can write into main memory when they should be writing to the display, thereby clobbering screen holes or program memory.

Keys in Menus

The MenuKey command gives your program the ability to use keypresses to select menu items. Typically, you use a combination keypress consisting of a letter key plus one of the Apple keys. Menu items that can be selected in this way are indicated by the OPEN-APPLE or SOLID-APPLE icon and the specified letter or other key displayed to the right of the menu item. If an item can be selected using either type of apple icon, the OPEN-APPLE icon appears with the letter in the menu.

You can also specify a control character as the keypress that selects a menu item. You do this by setting either Character 1 or Character 2 in the Menu Item Block to a value from 1 to 31, corresponding to a control character. (Menu Item Blocks are defined in Table 2-4.) You need not set the modifier bits in the Item Option Byte.

When you specify a control key to select an item, the Tool Kit displays a diamond icon and the key to the right of the menu item. Only the character in Character 1 will be used, even if you made Character 2 a control character.

Keypresses with the CONTROL key are easier to touch-type than those with the Apple keys, but you should still use the Apple-key combinations for most items and reserve the use of control keys for high-speed or repetitive functions where the ability to touch-type the command is important.

The user will expect control keys to be used for the same functions across different products. Apple has defined the menu functions of most of the control keys, as shown in Table 2-1.

If the user presses a key other than one of those specified in the menu, the Tool Kit generates a beep.

Table 2-1.
Control Keys for Menu Items

Control Key	Function
CTRL-B	Boldface
CTRL-C	Copy
CTRL-D	Delete
CTRL-E	Editing type, insert or overstrike cursor
CTRL-F	Forward delete
CTRL-H	Left arrow
CTRL-I	Tab
CTRL-J	Down arrow
CTRL-K	Up arrow
CTRL-L	Begin or end underline
CTRL-M	Return
CTRL-P	Print
CTRL-U	Right arrow
CTRL-V	Paste
CTRL-X	Cut
CTRL-Z	Zoom
CTRL-[	Escape

InitMenu

Function:

InitMenu establishes an area of memory that will be used to save the part of the display obscured by menus.

Command Number: 9 ($09)

Parameter List:

2 (input, byte) number of parameters

sa (input, word) save area: pointer to reserved memory area

sas (input, word) save area size: number of bytes reserved

Description:

During calls to MenuSelect, the part of the display obscured by a menu must be saved so that it can be replaced when the menu goes away. The application program must provide memory space and reserve it for use by the Tool Kit.

You can determine the amount of memory space to reserve for menu displays by calculating the screen area of the largest menu in the program. The largest menu could have a large screen area because it has many items, or it could have only a few items, each of which is very long.

You calculate the screen area of a menu by taking the product of the number of items in the menu, plus 1, times five bytes more than the length of the longest item string in that menu. If you are using keys to select items, each item string must include three bytes to display a space, an Apple icon, and the key that selects the item.

When the program calls the SetMenu command to initialize a menu bar, SetMenu checks whether the amount of memory reserved by InitMenu is enough for a particular menu and returns an error if it is not.

Error Codes:

(none)

SetMenu

Function:

SetMenu initializes the menu bar data structure and displays the menu bar.

Command Number: 10 ($0A)

Parameter List:

l	(input, byte) number of parameters
mbs	(input, word) pointer to menu bar structure

Description:

SetMenu initializes a menu bar data structure and displays the menu bar. Given a pointer to a menu bar structure (see Tables 2-2 and 2-4), SetMenu fills in the data required by the menu commands and saves the pointer for their use. Once SetMenu has been called, the program must not move the data structure.

SetMenu checks to make sure that the memory area reserved by InitMenu is enough to handle the display area that will be obscured by the menu bar specified by the data structure. If it is not, SetMenu returns an error, but it still displays the menu bar.

Error Codes:

10 ($0A)	Save area (from InitMenu) is too small.

Table 2-2. Data Structure for a Menu Bar

Parameter Function	Parameter Size	Note
Number of Menus	1 byte	
Reserved for Future Use	1 byte	
First Menu Block:		
Menu ID (can't be Ø)	1 byte	
Menu Option Byte	1 byte	
Pointer to Title String	2 bytes	
Pointer to Menu Data Structure	2 bytes	
X Position for Title Display	1 byte	*
Left for HiLite and Select	1 byte	*
Right for HiLite and Select	1 byte	*
Reserved for Future Use	1 byte	*
Second Menu Block:		
(same structure as First Menu Block)		
.		
.		
Last Menu Block		
(same structure as First Menu Block)		

*Indicates items filled in by Tool Kit.

Table 2-3. Contents of Option Byte in Each Menu Block (see Table 2-2)

Bit Number	Bit Function	Note
7	Disable Flag	*
6	Reserved for Future Use	
5	Reserved for Future Use	
4	Reserved for Tool Kit	
3	Reserved for Tool Kit	
2	Reserved for Future Use	
1	Reserved for Future Use	
Ø	Reserved for Future Use	

*Disable Flag is updated by DisableMenu command. By setting the flag

off before calling SetMenu, the program can make the menu start out disabled.

Table 2-4. Data Structure for a Menu

Parameter Function	Parameter Size	Note
Number of Items	1 byte	
Left Column of Save Box	1 byte	1
Right Column of Save Box	1 byte	1
Reserved for Future Use	1 byte	1
First Menu Item Block:		
Item Option Byte	1 byte	
Mark Character	1 byte	2
Character 1 (high bit off)	1 byte	3
Character 2 (high bit off)	1 byte	3
Pointer to Item String	2 bytes	
Second Menu Item Block:		
(Same structure as First Menu Item Block)		
.		
.		
Last Menu Item Block:		
(Same structure as First Menu Item Block)		

1 Indicates items filled in by Tool Kit.
2 Updated by the SetMark command. The program can set the initial mark character in the data structure, but after that it should change the mark character only by calling SetMark.
3 The program should set this byte to Ø if not using characters.

Table 2-5. Contents of Option Byte in Menu Data Structure

Bit Number	Bit Function	Notes
7	Disable Flag	1,5
6	Item Is Filler	2
5	Item Is Checked	3,5
4	Reserved for Tool Kit	
3	Reserved for Tool Kit	
2	Item Has Mark	3,5
1	Modifier Is SOLID-APPLE Key	
0	Modifier Is OPEN-APPLE Key	

1 Updated by the DisableItem Command.

2 If the "Item Is Filler" bit in the Option Byte is on, then Character 1 of the Menu Item Block (see Table 2-4) is the character to use for filler; otherwise, Character 1 and Character 2 are the uppercase and lowercase values of the key that identifies the item when MenuKey is called.

3 Updated by the CheckItem command.

4 Used only with Applesoft BASIC; set to 0 otherwise. See Appendix F, "Applesoft String Options."

5 The program can set the initial states of these flags in the data structure before calling the SetMenu command. After that, the program should update the flags only by calling the appropriate commands.

MenuSelect

Function:

MenuSelect interacts with the mouse to display a menu and return the selection, if any.

Command Number: 11 ($0B)

Parameter List:

2	(input, byte) number of parameters
id	(output, byte) menu ID, 0 = no menu item chosen
in	(output, byte) menu item number, undefined if id = 0

Description:

MenuSelect performs the interactive display of menus while the user keeps the mouse button depressed. MenuSelect does not return until the user releases the button and a button-up event occurs.

The application program calls MenuSelect whenever the user presses the mouse button on line 0 of the display. As the user moves the mouse up and down the menu display, MenuSelect tracks the mouse and updates the cursor. When the cursor moves onto a menu item, MenuSelect highlights the name of the item.

When the user releases the mouse button while the cursor is on a menu item, MenuSelect removes the menu from the display, highlights the menu title, and returns the menu ID number and the item number. After the program finishes performing the selected operation, it must call HiLiteMenu to turn off the highlighting of the menu title.

An application can also use the MenuSelect command in keyboard mouse emulation mode by calling it immediately after calling the KeyboardMouse command. In that mode, the Tool Kit tracks the cursor while the user presses cursor keys to move the cursor. The user indicates a menu selection by pressing the RETURN key or by pressing and releasing the mouse button. The user can also press an appropriate command key. Pressing the ESC key terminates the command.

Error Codes:

(none)

MenuKey

Function:

MenuKey finds the menu item that matches a key.

Command Number: 12 ($0C)

Parameter List:

4	(input, byte) number of parameters
id	(output, byte) menu ID, 0 if no item selected
in	(output, byte) item number, undefined if id = 0
k	(input, byte) key: the character typed
km	(input, byte) key modifier, as returned by GetEvent: 0 = no modifier, 1 = OPEN-APPLE key 2 = SOLID-APPLE key 3 = either Apple key

Description:

After the user presses a key, MenuKey searches the menu data to find a menu item that has a matching key. If it finds a match, it highlights the menu title and returns the menu ID number and item number the same way MenuSelect does.

Also like MenuSelect, MenuKey leaves the selected menu highlighted; the progam must call HiLiteMenu to turn off the highlighting.

If you set the key modifier parameter to 3, either Apple key will serve to modify a matching keypress.

If an item is disabled, its menu key or keys will not select it.

As a special case, the MenuKey command can operate like MenuSelect does in keyboard mouse emulation mode. Calling MenuKey with ESC as the key initiates that mode of operation. The Tool Kit tracks the cursor while the user presses cursor keys to move the cursor. The user indicates his selection by pressing the RETURN key or pressing and releasing the mouse button. The user can also press an appropriate command key. Pressing the ESC key terminates the command.

Error Codes:

(none)

HiLiteMenu

Function:

HiLiteMenu turns highlighting of a menu title on or off.

Command Number: 13 ($ØD)

Parameter List:

l	(input, byte)	number of parameters
id	(input, byte)	menu ID: Ø = turn highlighting off

Description:

HiLiteMenu turns highlighting of a specified menu title in the Menu Bar on. Call HiLiteMenu with id = Ø to turn off highlighting after a call to MenuSelect or MenuKey.

Error Codes:

8 ($Ø8) Menu ID was not found.

DisableMenu

Function:

DisableMenu disables or enables selection and highlighting over a whole menu.

Command Number: 14 ($0E)

Parameter List:

2 (input, byte) number of parameters

id (input, byte) menu ID

dis (input, byte) disable:
1 = disable
0 = enable

Description:

DisableMenu disables or enables selection and highlighting over a whole menu. If the menu has been disabled, none of the items can be selected, either by MenuSelect or by MenuKey. The menu will still appear when the user moves the mouse onto the menu title, but the title will not be highlighted, and none of the items in the menu will be highlighted when the mouse moves onto them.

When a call to DisableMenu enables a menu, any items that were individually disabled remain disabled. (See the DisableItem command.)

By setting the Disable Flag in the Menu Block's Option Byte when you set up the Menu Bar data structure, your program can make the menu start out disabled. After that, the program should use the DisableMenu command to disable and enable menus.

Error Codes:

8 ($08) Menu ID was not found

DisableItem

Function:

DisableItem disables or enables selection and highlighting of a menu item.

Command Number: 15 ($ØF)

Parameter List:

3	(input, byte) number of parameters
id	(input, byte) menu ID
in	(input, byte) item number
dis	(input, byte) disable: 1 = disable item Ø = enable item

Description:

DisableItem disables or enables selection and highlighting of a menu item. If an item is disabled, it cannot be selected, either by MenuSelect or by MenuKey, and it will not be highlighted when the mouse moves onto it.

To enable an item, call DisableItem with the disable parameter set to Ø.

By setting the Disable Flag in the Menu Item Block's Item Option Byte when you set up the menu data structure, your program can make the menu item start out disabled. After that, the program should use the DisableItem command to disable and enable menu items.

Calling DisableItem with item number set to zero generates error 9, Item Number Not Valid.

Error Codes:

8 ($Ø8)	Menu ID was not found
9 ($Ø9)	Item Number not valid

CheckItem

Function:

CheckItem turns the checkmark displayed next to item on or off.

Command Number: 16 ($10)

Parameter List:

3 (input, byte) number of parameters

id (input, byte) menu ID

in (input, byte) item number

ck (input, byte) checkmark:
Ø = turn checkmark off
1 = turn checkmark on

Description:

CheckItem turns the checkmark displayed next to item on or off. The checkmark appears in the blank column in the left edge of the menu.

Your program can call the SetMark command to change the checkmark to any ASCII character.

Calling CheckItem with item number set to zero generates error 9, Item Number Not Valid.

Error Codes:

8 ($Ø8) Menu ID was not found

9 ($Ø9) Item Number not valid

SetMark

Function:

SetMark enables a program to select the character to display for items that are checked in a menu.

Command Number: 20 ($14)

Parameter List:

4	(input, byte) number of parameters
id	(input, byte) menu ID
in	(input, byte) item number
mk	(input, byte) checkmark: 0 = use checkmark character 1 = install new character
char	(input, byte) character to display for this item

Description:

SetMark sets the character that is displayed when a program calls CheckItem. The default character is the checkmark.

Error Codes:

8 ($08)	Menu ID was not found
9 ($09)	Item Number not valid

Window Commands

The Tool Kit's window management commands provide the functions your application program needs to set up and display windows. Once you have set up the window information structure with OpenWindow, you can use these commands to select a window, bring it to the front of the display, put text into it, drag it, change its size, or close it.

Each open window must have a unique ID number in the range from 1 through 255. An attempt to open a second window with the same ID number as one already open will return an error.

A window ID number of Ø is not valid because FrontWindow returns ID = Ø when no window is open. An attempt to open a window with ID number of Ø will return an error.

With some of the Tool Kit commands, you can use an ID number of Ø to indicate the front window. If there is no front window when you do this, these commands return an error. The commands that interpret ID = Ø to mean the front window are

- CloseWindow
- GetWinPtr
- SelectWindow
- DragWindow
- GrowWindow
- WindowToScreen
- ScreenToWindow
- WinChar
- WinString
- WinText
- WinBlock
- WinOp

Note: The use of ID = Ø to select the front window is only a convenience. You can always use the actual ID number of the front window instead.

InitWindowMgr

Function:

InitWindowMgr initializes the internal list of open windows and establishes an area of memory that will be used to save parts of the display while a window is being dragged or grown.

Command Number: 22 ($16)

Parameter List:

2	(input, byte) number of parameters
ptr	(input, word) pointer to reserved memory area
size	(input, word) size of reserved memory area, in bytes

Description:

InitWindowMgr resets the pointers to the first and last entries in the internal linked list of open windows and establishes an area of memory that will be used to save parts of the display while a window is being dragged or grown.

During calls to DragWindow and GrowWindow, the Tool Kit must save the part of the display obscured by the outline of the window so that it can be replaced when the window operation is finished. The application program must provide the necessary memory space and reserve it for use by the Tool Kit.

The amount of memory space required is determined by the perimeter of the largest window (the sum of twice the window's width plus twice its length).

Note: This memory area can be the same as the area reserved by InitMenu.

Error Codes:

(none)

OpenWindow

Function:

OpenWindow opens a window by supplying a pointer to the window's Information Data Structure.

Command Number: 23 ($17)

Parameter List:

1	(input, byte) number of parameters
ptr	(input, word) pointer to Window Information Data Structure

Description:

OpenWindow passes window information to the Tool Kit via a pointer to a Window Information Data Structure, or Winfo Data Structure (see Table 2-6). The Winfo Data Structure must reside at a fixed location in memory while the window is open.

The Window Information Data Structure includes a pointer to a Document Information Data Structure (Dinfo Data Structure) that the Tool Kit uses to obtain the text to display in the window (see Table 2-10). Each call to OpenWindow makes that window the front, or active, window.

OpenWindow forces X and Y position coordinates to valid values. It also forces the Thumb positions to be no greater than the maximums. However, OpenWindow does not check to make sure that window minimums are less than maximums or that current window size is between the maximum and the minimum.

The application program can substitute its own routine for OpenWindow. The program passes the address of its open routine in the Winfo Data Structure in place of the pointer to the Dinfo Data Structure and set bit 7 of the Window Option Byte. The Tool Kit will pass control to the program's routine whenever the contents of the window need to be changed.

Because the user routine is called from within the Tool Kit, it cannot rely on the zero-page locations the Tool Kit uses (currently $00 to $18). When the Tool Kit calls the user routine, the register contents are

- accumulator: window ID number
- X register: low byte of Winfo address
- Y register: high byte of Winfo address

The routine can only call the Tool Kit commands whose names start with Win- to update the content region of the window it was requested to update. Any other calls can put the Tool Kit into an unknown state.

Error Codes:

12 ($0C) A window with the same ID is already open

13 ($0D) InitWindowMgr buffer too small for this window

14 ($0E) Bad Winfo—tried to open with ID=0, or conflicting maximum and minimum width or length

17 ($11) Error returned by user hook

Table 2-6. Information Structure for a Window

Parameter Function	Parameter Size	Note
Window ID Number (not 0)	1 byte	
Window Option Byte	1 byte	
Title String Pointer	2 bytes	
Window Position X Coordinate	2 bytes	1,2
Window Position Y Coordinate	2 bytes	1,2
Current Content Width	1 byte	1,3
Current Content Length	1 byte	1,3
Minimum Content Width	1 byte	
Maximum Content Width	1 byte	4
Minimum Content Length	1 byte	
Maximum Content Length	1 byte	4
Document Information Structure Pointer	2 bytes	
Horizontal Control Option Byte	1 byte	
Vertical Control Option Byte	1 byte	
Horizontal Scroll Maximum	1 byte	
Current Horizontal Thumb Position	1 byte	1,5
Vertical Scroll Maximum	1 byte	
Current Vertical Thumb Position	1 byte	1,5
Window Status Byte	1 byte	1
Reserved for Future Use	1 byte	6
Pointer to Next Winfo Structure	2 bytes	6
Reserved for Tool Kit	2 bytes	6
Screen Area Covered	4 bytes	6

1 Program sets initial values, Tool Kit updates these.
2 Initial values determine initial position of window.
3 Initial values determine initial window size.
4 Document width and length determine maximum content width and length.
5 Initial values determine initial position of thumb.
6 Items filled in by Tool Kit.

Table 2-7. Contents of Window Option Byte in Window Information Structure

Bit Number	Bit Function	Notes
7	Document Pointer Function	1
6	Reserved for Future Use	
5	Reserved for Future Use	
4	Reserved for Tool Kit	2
3	Reserved for Tool Kit	2
2	Grow Box is present	3
1	Close Box is present	3
Ø	Window is Dialog or Alert Box	3

1 This bit indicates the function of the Document Pointer.
Ø = Pointer to Document Information Structure
1 = Pointer to User Window Routine
2 The program must set these bytes to Ø.
3 These items set the initial appearance of the window. They cannot be changed when the window is open; instead, you must close the window, change the values, then open the window again.

Table 2-8. Contents of Horizontal or Vertical Control Option Byte in Window Information Structure

Bit Number	Bit Function	Notes
7	Scrollbar is present	1
6	Thumb is present	1
5	Reserved for Future Use	
4	Reserved for Future Use	
3	Reserved for Future Use	
2	Reserved for Future Use	
1	Reserved for Future Use	
0	Scrollbar is active	2

1 These items set the initial appearance of the window. They cannot be changed when the window is open; instead, you must close the window, change the values, then open the window again.
2 Initial value set by program; after that, use ActicateCtl to change it.

Table 2-9. Contents of Window Status Byte in Window Information Structure

Bit Number	Bit Function	Note
7	Window is open	*
6	Reserved for Future Use	
5	Reserved for Future Use	
4	Reserved for Future Use	
3	Used by Tool Kit	
2	Used by Tool Kit	
1	Used by Tool Kit	
0	Used by Tool Kit	

* Program can read to determine state of window.

Table 2-10. Information Structure for a Document

Parameter Function	Parameter Size	Note
Document Pointer	2 bytes	1
Reserved (set to Ø)	1 byte	
Document Width	1 byte	
Document X Coordinate	2 bytes	2
Document Y Coordinate	2 bytes	2
Reserved for Tool Kit	4 bytes	3

1 See bit 7 of the Window Option Byte
2 Set to Ø or set initial position in the document.
3 The program must set these bytes to Ø.

CloseWindow

Function:

CloseWindow removes the window with given ID number and redisplays the screen.

Command Number: 24 ($18)

Parameter List:

l	(input, byte) number of parameters
id	(input, byte) ID number of window to close

Description:

CloseWindow removes the window with the given ID number from the list of open windows and redisplays the screen with the window removed. Setting ID = Ø selects the top window as the window to be closed.

Error Codes:

15 ($ØF)	Window ID not found
17 ($11)	Error returned by user hook

CloseAll

Function:

CloseAll closes all open windows and redisplays the screen.

Command Number: 25 ($19)

Parameter List:

Ø	(input, byte) number of parameters

Description:

CloseAll removes all windows from the list of open windows and redisplays the screen.

Error Codes:

(none)

GetWinPtr

Function:

GetWinPtr returns the pointer to the Winfo structure of the open window that has the specified ID number.

Command Number: 45 ($2D)

Parameter List:

2 (input, byte) number of parameters

id (input, byte) ID number of window

ptr (output, word) pointer to Winfo Data Structure

Description:

GetWinPtr returns the pointer to the Window Information Data Structure (Winfo) of the open window that has the specified ID number. Setting ID = Ø selects the top window.

Error Codes:

15 ($ØF) Window ID not found

FindWindow

Function:

FindWindow returns the ID number of the window that contains the given point.

Command Number: 26 ($1A)

Parameter List:

4 (input, byte) number of parameters

px (input, byte) X mouse-coordinate of point

py (input, byte) Y mouse-coordinate of point

type (output, byte) type of area point is in:
Ø = Desktop
1 = Menu Bar
2 = content region
3 = drag region
4 = Grow Box
5 = Close Box

id (output, byte) ID number of window the point is in (Ø if point is in desktop or menu bar).

Description:

FindWindow returns the ID number of the window that contains the given point and returns the type of region the point is in: Menu Bar, content region, drag region, Grow Box, or Close Box. The point is specified in mouse coordinates. If the point is not in a window, FindWindow returns an ID number of Ø and a region type of desktop.

If the point is in the content region, the application program should call the FindControl command with window coordinates of the point to determine whether the point is in a scroll bar.

Error Codes:

(none)

FrontWindow

Function:

FrontWindow returns the ID number of the front window.

Command Number: 27 ($1B)

Parameter List:

l	(input, byte) number of parameters
id	(output, byte) ID number of front window

Description:

FrontWindow returns the ID number of the front, or active, window. It returns Ø if no windows are open.

Error Codes:

(none)

SelectWindow

Function:

SelectWindow activates the window with the given ID number.

Command Number: 28 ($1C)

Parameter List:

i	(input, byte) number of parameters
id	(input, byte) ID number of window

Description:

SelectWindow makes the window with the given ID number the front, or active, window and redisplays the screen. The window that was active becomes the second window in the list. Setting ID = Ø selects the front window. If the window selected is already the front window, the Tool Kit does not redisplay the screen.

Error Codes:

15 ($ØF)	Window ID not found
17 ($11)	Error returned by user hook

TrackGoAway

Function:

TrackGoAway tracksthe mouse and indicates whether the mouse button was released in the Go-Away Box.

Command Number: 29 ($1D)

Parameter List:

l	(input, byte) number of parameters
go	(output, byte) Go-Away status: Ø = not in Go-Away Box 1 = mouse was in Go-Away Box

Description:

TrackGoAway tracks the mouse until the mouse button is released. If the mouse is in the Go-Away Box when the button is released, the return status is 1; if not, it is Ø.

The application program should call TrackGoAway when it detects that the mouse button is down with the mouse in the Go-Away Box of the front window. If the return status indicates that the button was released in the Go-Away Box, the application program should then call the CloseWindow command.

Error Codes:

16 ($1Ø) There are no windows

DragWindow

Function:

DragWindow displays the outline of the window being dragged, then redisplays it in its new position.

Command Number: 30 ($1E)

Parameter List:

3	(input, byte) number of parameters
id	(input, byte) ID number of window being dragged
mx	(input, byte) X mouse coordinate of starting position
my	(input, byte) Y mouse coordinate of starting position

Description:

DragWindow displays the outline of the window being dragged until the user releases the mouse button, whereupon DragWindow clears the display area previously occupied by the window and redisplays the windows from back to front.

The application program should call the DragWindow command when it detects that the mouse button is down in the drag region of a window. In addition to the ID number of the window, the DragWindow command also needs the mouse coordinates of the position returned as px and py by the FindWindow command. In this it differs from the TrackGoAway and GrowWindow commands; while the Go-Away Box and the Grow Box consist of only one character each, the drag bar consists of several characters, and the mouse could be in any of them when the user starts dragging the window.

Setting ID = 0 selects the front window.

An application can also use the DragWindow command in keyboard mouse emulation mode by calling it immediately after calling the KeyboardMouse command. In that mode, the Tool Kit tracks the cursor and moves the window outline while the user presses cursor keys. The user indicates the completion of the move by pressing the RETURN key or by pressing and releasing the mouse button. Pressing the ESC key terminates the command and redisplays the window in its original position.

Error Codes:

15 ($ØF) Window ID not found

17 ($11) Error returned by user hook

22 ($16) Operation cannot be performed

GrowWindow

Function:

GrowWindow displays the outline of the window being grown, then redisplays an empty window with the new size.

Command Number: 31 ($1F)

Parameter List:

l	(input, byte) number of parameters
stat	(output, byte) return status: 0 = window did not change size 1 = window did change size

Description:

GrowWindow displays the outline of the window being grown until the user releases the mouse button, whereupon GrowWindow clears the display area previously occupied by the window and redisplays the windows from back to front.

The application program should call the GrowWindow command when it detects that the mouse button is down in the Grow Box of the front window.

GrowWindow leaves the content area of the front window blank because it can't determine whether the bottom of the document has been passed and whether the content area should be shifted. If the return status indicates that GrowWindow changed the size of the window, the application must redisplay the content area and update the scroll bars.

An application can also use the GrowWindow command in keyboard mouse emulation mode by calling it immediately after calling the KeyboardMouse command. In that mode, the Tool Kit tracks the cursor and draws the window outline in diffferent sizes while the user presses cursor keys. The user indicates the completion of the resizing by pressing the RETURN key or by pressing and releasing the mouse button. Pressing the ESC key terminates the command and redisplays the window in its original size.

Error Codes:

16 ($1Ø) There are no windows

17 ($11) Error returned by user hook

22 ($16) Operation cannot be performed

WindowToScreen

Function:

WindowToScreen converts window coordinate values to screen coordinates.

Command Number: 32 ($2Ø)

Parameter List:

5	(input, byte) number of parameters
id	(input, byte) ID number of window to use
wx	(input, word) X coordinate in the window
wy	(input, word) Y coordinate in the window
sx	(output, word) X coordinate for the screen
sy	(output, word) Y coordinate for the screen

Description:

WindowToScreen converts passed coordinate values from window coordinates to screen coordinates.

Setting ID = Ø selects the front window.

Error Codes:

15 ($ØF) Window ID not found

ScreenToWindow

Function:

ScreenToWindow converts screen coordinate values to window coordinates.

Command Number: 33 ($21)

Parameter List:

5 (input, byte) number of parameters

id (input, byte) ID number of window to use

sx (input, word) X coordinate for the screen

sy (input, word) Y coordinate for the screen

wx (output, word) X coordinate in the window

wy (output, word) Y coordinate in the window

Description:

ScreenToWindow converts passed coordinate values from screen coordinates to window coordinates.

Setting ID = Ø selects the front window.

Error Codes:

15 ($ØF) Window ID not found

WinChar

Function:

WinChar writes a character in a window.

Command Number: 34 ($22)

Parameter List:

4	(input, byte) number of parameters
id	(input, byte) ID number of window
wx	(input, word) X coordinate in window
wy	(input, word) Y coordinate in window
char	(input, byte) character to display

Description:

WinChar writes a character at a given position in a window. If the position given is not inside the window, WinChar does not write the character.

WinChar does not update the document.

Setting ID = 0 selects the front window.

Error Codes:

15 ($0F) Window ID not found

WinString

Function:

WinString writes a string in a window.

Command Number: 35 ($23)

Parameter List:

5 (input, byte) number of parameters

id (input, byte) ID number of window

wx (input, word) X coordinate in window

wy (input, word) Y coordinate in window

ptr (input, word) pointer to the string

res (input, byte) must be Ø.

Description:

WinString writes a string at a given position in a window. WinString does not wrap around; if the string extends past the right edge of the window, WinString just truncates it. WinString does not display any characters in the string that fall outside the edges of the window.

WinString does not update the document.

Setting ID = Ø selects the front window.

Error Codes:

15 ($ØF) Window ID not found

WinText

Function:

WinText writes ASCII characters in a window.

Command Number: 38 ($26)

Parameter List:

5	(input, byte) number of parameters
id	(input, byte) ID number of window
wx	(input, word) X coordinate in window
wy	(input, word) Y coordinate in window
ptr	(input, word) pointer to the first character of text
len	(input, byte) number of characters to display

Description:

WinText writes ASCII characters at a given position in a window. WinText does not wrap around; if the characters extend past the right edge of the window, WinText just truncates them. WinText does not display any characters that fall outside the edges of the window.

WinText does not update the document.

Setting ID = Ø selects the front window.

Error Codes:

15 ($ØF) Window ID not found

WinBlock

Function:

WinBlock writes a block of text in a window.

Command Number: 36 ($24)

Parameter List:

6 (input, byte) number of parameters

id (input, byte) ID number of window

ptr (input, word) pointer to Document Information Data Structure for the text to be displayed. If ptr = Ø, WinBlock uses the Dinfo pointer from the Winfo specified by the window ID. (Document Information Data Structure definition is Table 2-9.)

startx (input, word) X coordinate of upper-left corner of display window position within the document window (see Fig. 1-4)

starty (input, word) Y coordinate of upper-left corner of display window position within the document window (see Fig. 1-4)

stopx (input, word) X coordinate of lower-right corner of display window position within the document window (see Fig. 1-4)

stopy (input, word) Y coordinate of lower-right corner of display window position within the document window (see Fig. 1-4)

Description:

WinBlock writes a block of text in a window. Startx, starty, stopx, and stopy define a rectangle in the window where the characters are displayed; WinBlock does not alter anything outside this rectangle.

WinBlock does not update the document.

Setting ID = Ø selects the front window.

Error Codes:

15 ($ØF) Window ID not found

WinOp

Function:

WinOp performs an operation on a window.

Command Number: 37 ($25)

Parameter List:

4 (input, byte) number of parameters

id (input, byte) ID number of the window

wx (input, word) X window coordinate

wy (input, word) Y window coordinate

op (input, byte) operation to perform:

26 ($1A) = clear to start of window*
27 ($1B) = clear to start of line*
28 ($1C) = clear window
29 ($1D) = clear to end of window
30 ($1E) = clear line
31 ($1F) = clear to end of line

* Operations do not clear the character at position X,Y.

Description:

WinOp clears all or a portion of a window, depending on the operation code. Except for operation code 28, clear window, WinOp clears the characters from position X,Y to the end of the area indicated by the operation. Notice that the forward clears include the character at position X,Y, but the backward clears—that is, clear to start of window and clear to start of line—do not. You can think of the latter operations as "clear from start of area up to, but not through, position X,Y."

Setting ID = 0 selects the front window.

Error Codes:

15 ($0F) Window ID not found

Control Region Commands

These commands deal with the control regions in the front window: the horizontal and vertical scroll bars, including the Thumbs.

FindControl

Function:

FindControl indicates which control region of a window a given point is in.

Command Number: 39 ($27)

Parameter List:

4 (input, byte) number of parameters

wx (input, word) X window coordinate of point

wy (input, word) Y window coordinate of point

ctl (output, byte) control region the point is in:
- 0 = content region
- 1 = vertical scroll bar
- 2 = horizontal scroll bar
- 3 = none of the above (dead zone)

part (output, byte) part of control region the point is in:
- 1 = Up-Arrow of vertical scroll bar, Left-Arrow of horizontal scroll bar
- 2 = Down-Arrow of vertical scroll bar, Right-Arrow of horizontal scroll bar
- 3 = page-up region of vertical scroll bar, page-left region of horizontal scroll bar
- 4 = page-down region of vertical scroll bar, page-right region of horizontal scroll bar
- 5 = Thumb of scroll bar

Description:

FindControl indicates which control region of a window a given point is in. The application program should call FindControl when it determines, by means of a call to FindWindow, that the mouse is in the content region of the front window. Depending on the control and part codes returned by FindControl, the application should then take

appropriate action—for example, if the mouse is in a page-up or page-down region or in an Up-Arrow or Down-Arrow, the application scrolls the contents of the window, then calls UpdateThumb to make the Thumb reflect the new position in the file.

The application program must make sure that the wx and wy values are converted to window coordinates before calling FindControl.

Note: This is different from FindWindow, which takes mouse coordinates.

Error Codes:

16 ($10) There are no windows

SetCtlMax

Function:

SetCtlMax changes the range of the scroll bar of the front window.

Command Number: 40 ($28)

Parameter List:

2 (input, byte) number of parameters

ctl (input, byte) control region to update max value for:
1 = vertical scroll bar
2 = horizontal scroll bar

max (input, byte) new maximum value (must be greater than 1)

Description:

SetCtlMax changes the range of the scroll bar of the front window. If the current Thumb position is greater than the new maximum, SetCtlMax sets the Thumb to the new maximum and calls UpdateThumb to display it at the proper position. SetCtlMax changes the control max value and (if necessary) Thumb position in the Winfo Data Structure.

The program normally calls SetCtlMax whenever the size of a window changes (for example, by GrowWindow).

Maximum values depend on the application; a typical maximum value for the horizontal scroll bar would be calculated as the document width, minus the content width, plus twice the width of the vertical scroll bar or grow box. Likewise, a typical maximum value for the vertical scroll bar would be calculated as the document length, minus the content length, plus the height of the horizontal scroll bar.

Error Codes:

16 ($10) There are no windows

18 ($12) Bad control ID (not 1 or 2)

TrackThumb

Function:

TrackThumb tracks the thumb until the mouse button is released, then it updates the data in the Winfo.

Command Number: 41 ($29)

Parameter List:

3 (input, byte) number of parameters

ctl (input, byte) the control region whose Thumb is moving:
1 = vertical scroll bar
2 = horizontal scroll bar

pos (output, byte) position the Thumb moved to

stat (output, byte) return status:
Ø = Thumb didn't move, pos is not valid
1 = Thumb did move

Description:

TrackThumb tracks the Thumb until the mouse button is released. The application program should call TrackThumb when FindControl indicates that the mouse button is down in the Thumb. When the mouse button is released, TrackThumb updates the position information in the Winfo Data Structure and returns the new position of the Thumb. If the value of the return status is Ø, the Thumb is in the same position it started in, and the value of pos is not valid.

The Thumb position is a number in the range from Ø to the maximum position on the horizontal or vertical scrolling bar. A position of Ø means the first character of the document should be made visible; a position equal to the maximum means the last character of the document should be made visible.

If the Thumb position is the same as it was when TrackThumb is called, it is treated as if it had not moved. If the Thumb does move, TrackThumb updates the Thumb position in the Winfo Data Structure.

TrackThumb operates only on the front window.

Error Codes:

16 ($1Ø) There are no windows

18 ($12) Bad control ID (not 1 or 2)

UpdateThumb

Function:

UpdateThumb redisplays the Thumb in the designated position.

Command Number: 42 ($2A)

Parameter List:

2 (input, byte) number of parameters

ctl (input, byte) control region whose Thumb is being moved

pos (input, byte) new position of Thumb

Description:

UpdateThumb redisplays the Thumb in the designated position and updates the position value in the Winfo Data Structure. UpdateThumb operates only on the front window.

The program should call UpdateThumb after scrolling or paging.

Error Codes:

16 ($10) There are no windows

18 ($12) Bad control ID (not 1 or 2)

ActivateCtl

Function:

ActivateCtl changes the state of a scroll bar.

Command Number: 43 ($2B)

Parameter List:

2 (input, byte) number of parameters

ctl (input, byte) which control region to change

state (input, byte) state to make control region:
Ø = inactive
1 = active

Description:

ActivateCtl changes the state of a scroll bar and updates the Control Option Byte in the Winfo Data Structure. An active scroll bar shows the Thumb and page regions; an inactive bar shows a hollow page region.

ActivateCtl operates only on the front window.

Error Codes:

16 ($1Ø) There are no windows

18 ($12) Bad control ID (not 1 or 2)

Chapter 3

The Machine Language Interface

This chapter tells you what you need to know to use the MouseText Tool Kit with your machine language programs. For descriptions of the individual Tool Kit commands, see Chapter 2.

Installing the Machine Language Tool Kit

The MouseText Tool Kit is a relocatable package of machine language subroutines. To use the Tool Kit in a Pascal environment, you'll need to load the routines as a library unit, as described in Chapter 4. To use the Tool Kit in a ProDOS environment, you'll need to load the routines with the relocating loader that comes with the ProDOS Assembly Tools.

By The Way: Make sure you get up-to-date documentation and code for Rload; the early versions don't work.

Version 2.1 of the MouseText Tool Kit is designed to run only on the Apple IIe and only in the primary 64K memory space; later versions may be able to take advantage of auxiliary memory.

Important: Version 2.1 of the MouseText Tool Kit must reside in the main 64K memory bank, and all calls must be made from main memory.

All calls to the MouseText Tool Kit go through a single entry point named ToolKit. In addition to necessary housekeeping functions, the

main entry point of the MouseText Tool Kit saves the X and Y index registers and saves the locations in zero page that it uses for temporary storage.

The exit routine for the Tool Kit also performs housekeeping functions, as well as restoring the contents of the zero-page locations, restoring the previous contents of the X and Y index registers, and setting the carry flag to reflect the error status. The exit routine also loads the error status into the accumulator, thereby setting the 6502's N and Z flags.

Syntax of Machine Language Calls

A machine language call to the MouseText Tool Kit looks like this:

```
JSR     TOOLKIT         ;main Tool Kit entry point
DB      CMDNUM          ;command number of
                        ;routine being called
DW      CMDLIST         ;pointer to parameter list
BNE     ERROR           ;optional error handling
```

For programming examples showing calls to the MouseText Tool Kit, refer to Appendix D.

By The Way: Calls to the MouseText Tool Kit have the same syntax as calls to the ProDOS Machine Language Interface, which is described in the ProDOS Technical Reference Manual.

After a return from a call to the Tool Kit, the value of the program counter is six bytes beyond the location of the calling JSR, and the accumulator contains the error code. The index registers and the stack pointer are unchanged. If the called routine generated an error, the carry bit is on and the zero bit is off; if it did not generate an error, the zero bit is on and the carry bit is off. Table 3-1 gives a summary of the return status for Tool Kit calls.

Table 3-1. Processor Status After Return from Tool Kit. A bit value of x means the bit is undefined.

	Processor Status Bits N	Z	C	D	V	Accumulator Contents	Program Counter
Successful Call	0	1	0	0	x	0	Calling JSR + 6
Unsuccessful Call	0	0	1	0	x	Error Code	Calling JSR + 6

The Machine Language Commands

Here are the command numbers and parameter lists for each of the Tool Kit commands.

Startup Commands

A program normally calls appropriate startup commands once to set up its operating environment.

StartDeskTop

```
StartDesktop  equ   0          ; command number

start.parms   db    6          ; parameter list for StartDeskTop
start.mid     db    0          ; machine id byte
start.msid    db    0          ; machine subid byte
start.opsys   db    $00        ; using ProDOS
start.slotn   db    $00        ; slot no. for mouse (0 = check all slots)
start.int     db    $01        ; using Interrupt Mode
start.col     db    $01        ; using 80 columns
```

StopDesktop

```
StopDesktop   equ   1          ; command number

stop.parms    db    0          ; parameter list for StopDeskTop
```

PascIntAdr

```
PascIntAdr    equ    17        ; command number

pasc.parms    db     1         ; parameter list for PascIntAdr
pasc.addr     dw     0         ; address of int handler
```

SetBasAdr

```
SetBasAdr     equ    18        ; command number

setb.parms    db     1         ; parameter list for SetBasAdr
setb.addr     dw     0         ; base address of string area
```

Version

```
Version       equ    19        ; command number

ver.parms     db     2         ; parameter list for Version
ver.ver       db     0         ; version number
ver.rev       db     0         ; revision number
```

SetUserHook

```
SetUserHook   equ    47        ; command number

shook.parms   db     2         ; parameter list for SetUserHook
shook.id      db     0         ; user's routine ID
shook.addr    db     0         ; starting address of user's routine
```

KeyboardMouse

```
KeyboardMouse equ    48        ; command number

kdbms.parms   db     0         ; parameter list for KeyboardMouse
```

Cursor Commands

These commands control the appearance of the cursor.

SetCursor

```
SetCursor     equ    2         ; command number

setc.parms    db     1         ; parameter list for SetCursor
setc.char     db     $00       ; character to use for cursor
```

ShowCursor

```
ShowCursor    equ    3          ; command number
showc.parms   db     0          ; parameter list for ShowCursor
```

HideCursor

```
HideCursor    equ    4          ; command number
hidec.parms   db     0          ; parameter list for HideCursor
```

ObscureCursor

```
ObscureCursor equ    44         ; command number
obscc.parms   db     0          ; parameter list for ObscureCursor
```

Event-Handling Commands

These commands deal with events in the event queue.

CheckEvents

```
CheckEvents   equ    5          ; command number
chke.parms    db     0          ; parameter list for CheckEvents
```

GetEvent

```
GetEvent      equ    6          ; command number

evt.parms     db     3          ; parameter list for GetEvent
evt.type      db     0          ; the event type
evt.eb1       db     0          ; event byte 1 (x or key)
evt.eb2       db     0          ; event byte 2 (y or modifier)

evt.x         equ    evt.eb1    ; x pos of mouse
evt.y         equ    evt.eb2    ; y pos of mouse
evt.key       equ    evt.eb1    ; key input by user
evt.keymod    equ    evt.eb2    ; modifier to key input by user
```

FlushEvents

```
FlushEvents   equ   7          ; command number

flshe.parms   db    0          ; parameter list for FlushEvents
```

SetKeyEvent

```
SetKeyEvent   equ   8          ; command number

setkey.parms  db    1          ; parameter list for SetKeyEvent
setkey.sk     db    0          ; set key event
```

PeekEvent

```
PeekEvent     equ   21         ; command number

pke.parms     db    3          ; parameter list for PeekEvent
pke.type      db    0          ; the event type
pke.eb1       db    0          ; event byte 1 (x or key)
pke.eb2       db    0          ; event byte 2 (y or modifier)

pke.x         equ   pke.eb1    ; x pos of mouse
pke.y         equ   pke.eb2    ; y pos of mouse
pke.key       equ   pke.eb1    ; key input by user
pke.keymod    equ   pke.eb2    ; modifier to key input by user
```

PostEvent

```
PostEvent     equ   46         ; command number

post.parms    db    3          ; parameter list for PostEvent
post.type     db    0          ; the event type
post.eb1      db    0          ; event byte 1 (x or key)
post.eb2      db    0          ; event byte 2 (y or modifier)

post.x        equ   post.eb1   ; x pos of mouse
post.y        equ   post.eb2   ; y pos of mouse
post.key      equ   post.eb1   ; key input by user
post.keymod   equ   post.eb2   ; modifier to key input by user
```

Menu Commands

These commands deal with menu selection and display.

InitMenu

```
InitMenu      equ    9          ; command number

im.parms      db     2          ; parameter list for InitMenu
im.sarea      dw     savearea   ; area to use for saving screen under menu
im.ssize      dw     savesize   ; size of save area
```

SetMenu

```
SetMenu       equ    10         ; command number

sm.parms      db     1          ; parameter list for SetMenu
sm.mbar       dw     mymenu     ; pointer to Menu Data Structure
```

MenuSelect

```
MenuSelect    equ    11         ; command number

ms.parms      db     2          ; parameter list for MenuSelect
ms.mid        db     0          ; menu ID returned
ms.item       db     0          ; item number returned
```

MenuKey

```
MenuKey       equ    12         ; command number

mkey.parms    db     4          ; parameter list for MenuKey
mkey.mid      db     0          ; menu ID returned
mkey.item     db     0          ; item number returned
mkey.key      db     0          ; key user typed
mkey.mod      db     0          ; modifier of key
```

HiliteMenu

```
HiliteMenu    equ    13         ; command number

hili.parms    db     1          ; parameter list for HiliteMenu
hili.mid      db     0          ; menu ID (0 for all)
```

DisableMenu

```
DisableMenu   equ    14         ; command number

dism.parms    db     2          ; parameter list for DisableMenu
dism.id       db     0          ; menu ID
dism.dis      db     0          ; disable code
```

DisableItem

```
DisableItem   equ    15        ; command number

ditm.parms    db     3         ; parameter list for DisableItem
ditm.id       db     0         ; menu ID
ditm.item     db     0         ; item number
ditm.dis      db     0         ; disable code
```

CheckItem

```
CheckItem     equ    16        ; command number

chki.parms    db     3         ; parameter list for CheckItem
chki.id       db     0         ; menu ID
chki.item     db     0         ; item number
chki.chk      db     0         ; checkmark code
```

SetMark

```
SetMark       equ    20        ; command number

setm.parms    db     4         ; parameter list for SetMark
setm.id       db     0         ; menu ID
setm.item     db     0         ; item number
setm.chk      db     0         ; checkmark code
setm.char     db     0         ; character to use as checkmark
```

Window Commands

These commands deal with window selection and display.

InitWindowMgr

```
InitWindowMgr equ    22        ; command number

iwm.parms     db     2         ; parameter list for InitWindowMgr
iwm.sarea     dw     savearea  ; area to use when saving window screen
iwm.ssize     dw     savesize  ; size of save area
```

OpenWindow

```
OpenWindow    equ    23        ; command number

open.perm     db     1         ; parameter list for OpenWindow
open.wind     dw     0         ; pointer to Winfo Data Structure
```

CloseWindow

```
CloseWindow     equ     24          ; command number

cw.parms        db      1           ; parameter list for CloseWindow
cw.id           db      0           ; ID number of window to close
```

CloseAll

```
CloseAll        equ     25          ; command number

cla.parms       db      0           ; parameter list for CloseAll
```

GetWinPtr

```
GetWinPtr       equ     45          ; command number

gwip.parms      db      2           ; parameter list for GetWinPtr
gwip.id         db      0           ; window ID number
gwip.winfo      dw      0           ; pointer to Winfo Data Structure
```

FindWindow

```
FindWindow      equ     26          ; command number

fdw.parms       db      4           ; parameter list for FindWindow
fdw.x           db      0           ; X coordinate of mouse
fdw.y           db      0           ; Y coordinate of mouse
fdw.type        db      0           ; type of region mouse is in
fdw.window      db      0           ; window ID number (0 = desktop)
```

FrontWindow

```
FrontWindow     equ     27          ; command number

frtw.parms      db      1           ; parameter list for FrontWindow
frtw.id         db      0           ; ID number of front window
```

SelectWindow

```
SelectWindow    equ     28          ; command number

selw.parms      db      1           ; parameter list for SelectWindow
selw.id         db      0           ; ID number of window
```

TrackGoAway

```
TrackGoAway   equ    29        ; command number

tga.parms     db     1         ; parameter list for TrackGoAway
tga.closeit   db     0         ; Go-Away status
```

DragWindow

```
DragWindow    equ    30        ; command number

dg.parms      db     3         ; parameter list for DragWindow
dg.id         db     0         ; window ID number
dg.x          db     0         ; x mouse coord of cursor start
dg.y          db     0         ; y mouse coord of cursor start
```

GrowWindow

```
GrowWindow    equ    31        ; command number

grow.parms    db     1         ; parameter list for GrowWindow
grow.result   db     0         ; return status
```

WindowToScreen

```
WindowToScreen equ   32        ; command number

w2s.parms     db     5         ; parameter list for WindowToScreen
w2s.id        db     0         ; window ID number
w2s.wx        dw     0         ; X coordinate in window
w2s.wy        dw     0         ; Y coordinate in window
w2s.sx        dw     0         ; X screen coordinate
w2s.sy        dw     0         ; Y screen coordinate
```

ScreenToWindow

```
ScreenToWindow equ   33        ; command number

s2w.parms     db     5         ; parameter list for ScreenToWindow
s2w.id        db     0         ; window ID number
s2w.sx        dw     0         ; X screen coordinate
s2w.sy        dw     0         ; Y screen coordinate
s2w.wx        dw     0         ; X coordinate in window
s2w.wy        dw     0         ; Y coordinate in window
```

WinChar

```
WinChar       equ     34        ; command number

wch.parms     db      4         ; parameter list for WinChar
wch.id        db      Ø         ; window ID number
wch.wx        dw      Ø         ; X coordinate in window
wch.wy        dw      Ø         ; Y coordinate in window
wch.char      db      $ØØ       ; ASCII character to display
```

WinString

```
WinString     equ     35        ; command number

wstr.parms    db      5         ; parameter list for WinString
wstr.id       db      Ø         ; window ID number
wstr.wx       dw      Ø         ; X coordinate in window
wstr.wy       dw      Ø         ; Y coordinate in window
wstr.ptr      dw      Ø         ; pointer to string
wstr.res      db      Ø         ; reserved (for BASIC only)
```

WinText

```
WinText       equ     38        ; command number

wtxt.parms    db      5         ; parameter list for WinString
wtxt.id       db      Ø         ; window ID number
wtxt.wx       dw      Ø         ; X coordinate in window
wtxt.wy       dw      Ø         ; Y coordinate in window
wtxt.ptr      dw      Ø         ; pointer to first character
wtxt.len      db      Ø         ; number of characters
```

WinBlock

```
WinBlock      equ     36        ; command number

wblk.parms    db      6         ; parameter list for WinBlock
wblk.id       db      Ø         ; window ID number
wblk.ptr      dw      Ø         ; pointer to Dinfo Data Structure
wblk.x1       dw      Ø         ; X upper-left window coordinate
wblk.y1       dw      Ø         ; Y upper-left window coordinate
wblk.x2       dw      Ø         ; X lower-right window coordinate
wblk.y2       dw      Ø         ; Y lower-right window coordinate
```

WinOp

```
WinOp          equ     37          ; command number

wop.parms      db      4           ; parameter list for WinBlock
wop.id         db      0           ; window ID number
wop.wx         dw      0           ; X window coordinate
wop.wy         dw      0           ; Y window coordinate
wop.op         db      0           ; window operation
```

Control Region Commands

These commands deal with the control regions in the front window: the horizontal and vertical scrolls bars, including the Thumbs.

FindControl

```
FindControl    equ     39          ; command number

findc.parms    db      4           ; parameter list for FindControl
findc.wx       dw      0           ; X window coordinate of point
findc.wy       dw      0           ; Y window coordinate of point
findc.ctl      db      0           ; control region point is in
findc.part     db      0           ; part of region point is in
```

SetCtlMax

```
SetCtlMax      equ     40          ; command number

setct.parms    db      2           ; parameter list for SetCtlMax
setct.ctl      db      0           ; control region affected
setct.newmax   db      0           ; new maximum value
```

TrackThumb

```
TrackThumb     equ     41          ; command number

tkthmb.parms   db      3           ; parameter list for TrackThumb
tkthmb.ctl     db      0           ; control region affected
tkthmb.pos     db      0           ; position Thumb moved to
tkthmb.moved   db      0           ; Thumb moved code
```

UpdateThumb

```
UpdateThumb    equ     42          ; command number

upt.parms      db      2           ; parameter list for UpdateThumb
```

```
upt.ctl       db     0          ; control region affected
upt.newpos    db     0          ; new position of Thumb
```

ActivateCtl

```
ActivateCtl   equ    43         ; command number

actl.parms    db     2          ; parameter list for ActivateCtl
actl.ctl      db     0          ; ctl region to change
actl.inact    db     0          ; inactivate code
```

Chapter 4

The Pascal Interface

The Pascal Interface for the MouseText Tool Kit is a Pascal intrinsic unit that provides the interface to the MouseText Tool Kit, Version 2.1. Each of the Tool Kit commands described in Chapter 2 is supported by one of the Pascal Interface procedures described in this chapter. In addition to the command procedures, there is a utility procedure for obtaining the address of a Pascal variable.

Installing the Pascal Interface

The Pascal Interface and the Tool Kit routines are supplied together in a linked object file named MTXKIT.CODE. To use the Tool Kit, you must install MTXKIT.CODE as a Unit in your System.library file. With the Tool Kit code in the system library, the application program can use the Tool Kit commands by including the statement "Uses MTXKIT;" after the heading.

Data Structures

This chapter presents the specifications of the data types and data structures used in the Pascal Tool Kit, including the Menu Data Structure, the Window Information Data Structure, and the Document Information Data Structure, as defined in Chapter 2.

Constants and Type Definitions

The following constants and data types are used in the Pascal Interface.

Constants

max_menus= 10 (A maximum of 10 menus is supported).
max_title_str= 20 (A maximum of 20 characters per menu title is supported).
max_item_str= 30 (A maximum of 30 characters per menu item name is supported).
max_num_items= 10 (A maximum of 10 menu items is supported).

The following event type values are provided as constants rather than as an enumerated type so that the user can define and handle his own events.

```
no_event = 0
button_down = 1
button_up = 2
key_down = 3
drag = 4
apple_key = 5
```

A single byte value is defined as:

```
byte = 0..255;
```

Event

An event is defined as:

```
type_event = packed record
      evt_kind : byte;
      char1 : byte;
      char2 : byte;
      reserve1 : byte;
      end;
```

where:

evt_kind is the event type value (see above under Constants).
char1 is event byte 1, X coordinate or key value.
char2 is event byte 2, Y coordinate or key modifier.
reserve1 is reserved for use by the Tool Kit.

Menu titles are defined as:

```
title_str = string[max_title_str];
```

Menu Item Names

Menu item names are defined as:

```
item_str = string[max_item_str];
```

Menu Item Blocks

A Menu item block is defined as:

```
menu_item = packed record

      open_apple : boolean;          {bit 0}
      solid_apple : boolean;
      item_has_mark : boolean;
      reserve2 : boolean;
      reserve3 : boolean;
      item_is_checked : boolean;
      item_is_filler : boolean;
      disable_flag : boolean;        {bit 7}

      mark_char : byte;

      char1 : byte;
      char2 : byte;

      item_str_ptr : ^item_str;

      end;
```

where:

The first 8 fields in the record are the bits in the Item Option Byte:

open_apple is on when the modifier is OPEN-APPLE key;
solid_apple is on when the modifier is SOLID-APPLE key;
item_has_mark is on when the item has mark;
reserve2, reserve3 are reserved for use by the Tool Kit;
item_is_checked is on when the Item Is Checked;
item_is_filler is on when the Item Is Filler;
disable_flag is the Disable Flag;

mark_char is the mark character;

char1 is Character 1;
char2 is Character 2;

item_str_ptr is Pointer to Item String;

Menu Data Structures

The Data Structure for a Menu is defined as:

```
menu_data = packed record
      num_items : byte;
      reserve1 : byte;
      reserve2 : byte;
      reserve3 : byte;
      items : packed array [1..max_num_items] of menu_item;
      end;
```

where:

num_items is the Number of Items;
reserve1, reserve2, reserve3 are reserved for use by the Tool Kit;
items is the array of Menu Item Blocks;

Menu Title Blocks

A Menu Title Block is defined as:

```
menu_title = packed record
      menu_id : byte;
      disabled : byte;
      title_ptr : ^title_str;
      m_data_ptr : ^menu_data;
      reserved : packed array [1..4] of byte;
      end;
```

where:

menu_id is the Menu ID;
disabled is the Disable Flag (only bit 7 can be used);
title_ptr is the Pointer to Title String;
m_data_ptr is the Pointer to Menu Data Structure;
reserved is reserved by the Tool Kit;

Menu Bars

The menu bar is defined as:

```
menu_bar = packed record
      num_menus : byte;
      reserved : byte;
      menus : array [1..max_menus] of menu_title;
      end;
```

where:

num_menus is the Number of Menus;
reserved is reserved for use by the Tool Kit;
menus is the array of Menu Blocks;

Window Information Data Structures

A Window Information Data Structure (Winfo) is defined as:

```
winfo = packed record
      window_id: byte;

      dialog: boolean;                {bit Ø}
      goawaybox: boolean;
      growbox: boolean;
      reserve1: boolean;
      reserve2: boolean;
      reserve3: boolean;
      reserve4: boolean;
      dinfo_or_user: boolean;         {bit 7}

      title_ptr: ^title_str;

      windowx: integer;
      windowy: integer;

      contwidth: byte;
      contlength: byte;

      mincontwidth: byte;
      maxcontwidth: byte;
      mincontlength: byte;
      maxcontlength: byte;

      dinfo_ptr: ^dinfo;

      hactive: boolean;               {bit Ø}
      reserve6: boolean;
      reserve7: boolean;
      reserve8: boolean;
      reserve9: boolean;
      reserv1Ø: boolean;
      hthumb: boolean;
      hscrollbar: boolean;            {bit 7}

      vactive: boolean;               {bit Ø}
      reserv11: boolean;
      reserv12: boolean;
      reserv13: boolean;
```

```
      reserv14: boolean;
      reserv15: boolean;
      vthumb: boolean;
      vscrollbar: boolean;          {bit 7}

      hthumbmax: byte;
      hthumbpos: byte;
      vthumbmax: byte;
      vthumbpos: byte;

      reserv16: boolean;
      reserv17: boolean;
      reserv18: boolean;
      reserv19: boolean;
      reserv20: boolean;
      reserv21: boolean;
      reserv22: boolean;
      win_open: boolean;

      reserv23: byte;

      nextwinfo = ^winfo

      reserv24: byte;
      reserv25: byte;
      reserv26: byte;
      reserv27: byte;
      reserv28: byte;
      reserv29: byte;

      end;
```

where:

window_id is the Window ID#

dialog is dialog/alert window flag
goawaybox is on when Go-Away Box present
growbox is on when Grow Box present
reserve1, reserve2, reserve3, reserve4
are all reserved by the Tool Kit

dinfo_or_user is user routine adr/dinfo ptr

title_ptr is Title Str ptr

windowx is Window Location X
windowy is Window Location Y

contwidth is Current Content Width
contlength is Current Content Length

```
mincontwidth is Min Content Width
maxcontwidth is Max Content Width
mincontlength is Min Content Length
maxcontlength is Max Content Length

dinfo_ptr is Dinfo Ptr

hactive is on when horizontal scrollbar active
reserve6, reserve7, reserve8, reserve9, reserv10
      are all reserved by the Tool Kit
hthumb is on when horizontal Thumb present
hscrollbar is on when horizontal scroll bar present

vactive is on when vertical scroll bar active
reserv11, reserv12, reserv13, reserv14, reserv15
      are all reserved by the Tool Kit
vthumb is on when vertical Thumb present
vscrollbar is on when vertical scroll bar present

hthumbmax is horizontal scroll maximum
hthumbpos is current horizontal Thumb position
vthumbmax is vertical scroll maximum
vthumbpos is current vertical Thumb position

reserv16, reserv17, reserv18, reserv19, reserv20,
reserv21, and reserv22
      are all reserved by the Tool Kit

win_open is window open

reserv23 is reserved by the Tool Kit

nextwinfo is the pointer to the next winfo structure

reserv24, reserv25, reserv26, reserv27
reserv28, and reserv29
      are all reserved by the Tool Kit
```

Document Information Data Structures

A Document Information Data Structure (Dinfo) is defined as:

```
dinfo = packed record

      doc_ptr: integer;

      reserved: byte;
      docwidth: byte;

      docx: integer;
      docy: integer;
```

```
      doclength: integer;
      reserve2: byte;
      reserve3: byte;

      end;
```

where:

doc_ptr is Document ptr

reserved is reserved by the Tool Kit
docwidth is Document Width

docx is Document X
docy is Document Y

doclength is Document Length
reserve2, reserve3 are reserved by the Tool Kit

Screen Region Types

The type of screen region is defined as:

```
type_area = (inDeskTop,
      inMenubar,
      inContent,
      inDrag,
      inGrow,
      inGoAway);
```

here each value is as returned by FindWindow:

inDeskTop is in desktop
inMenubar is in menu bar
inContent is in contentregion
inDrag is in drag region
inGrow is in Grow Box
inGoAway is in Go-Away Box

Control Region Types

The type of control region is defined as:

```
ctlarea = ( notctl,
      ver_scroll,
      hor_scroll,
      deadzone );
```

where each value is as returned by FindControl:

notctl is in content region
ver_scroll is in vertical scroll bar
hor_scroll is in horizontal scroll bar
deadzone is none of the above

Control Region Part Types

The type of a part of a control region is defined as:

```
ctlpart = ( ctlinactive,
      scrollupleft,
      scrolldownright,
      pageupleft,
      pagedownright,
      thumb );
```

where each value is as returned by FindControl:

ctlinactive is never returned
scrollupleft is up arrow of vertical scroll bar
      or Left-Arrow of horizontal scroll bar
scrolldownright is Down-Arrow of vertical scroll bar
      or Right-Arrow of horizontal scroll bar
pageupleft is "page up" region of vertical scroll bar
      or "page left" region of horizontal scroll bar
pagedownright is "page down" region of vertical scroll bar
      or "page right" region of horizontal scroll bar
thumb is Thumb of scroll bar

Pointers

A general purpose pointer is provided and defined as:

```
pointer: integer;
```

Error Codes

The Mouse Tool Kit error code is defined as:

```
TKError : integer;
```

Command Functions and Procedures

Here are the specifications of the procedure calls in the Pascal Tool Kit Interface.

Startup Commands

A program will normally call the appropriate startup commands once to set up the operating environment. The proper sequence of steps to start the mouse is:

(1) Call PascIntAdr to get the address of the Tool Kit's interrupt handler.

(2) Pass the interrupt address to the mouse firmware by calling SetMouse as described in Appendix B, "The Mouse Firmware Interface." Mouse Mode should be set to passive.

(3) Call StartDesktop with the UseInterrupts parameter set the way you want it for your program.

(4) (optional) Call SetUserHook to pass the addresses of your program's interrupt handlers, if any, to the Tool Kit.

StartDeskTop

```
Procedure StartDeskTop ( mach_id :  integer; sub_id:  integer;
      var slot_num :  integer; use_interrupts :  boolean;
      column_80 :  boolean );
```

mach_id is the machine ID number.
sub_id is the subsidiary ID number.
slot_num is the slot number of the mouse card.
use_interrupts is the interrupt usage parameter:
false= Passive Mode only
true= use interrupts
column_80 is the col (number of text columns) parameter:
false= 40 columns
true= 80 columns

StopDeskTop

```
Procedure StopDeskTop;
```

PascIntAdr

```
Procedure PascIntAdr ( var IntAdr:  integer );
```

IntAdr is the address of the interrupt routine

SetUserHook

```
Procedure SetUserHook ( hook_id, hook_adr: integer );
```

hook_id is the ID number (0 or 1) for the program's interrupt routine.
hook_adr is the address of the program's interrupt routine.

Version

```
Procedure Version ( var ver_num, rev_num:  integer);
```

ver_num is the version number.
rev_num is the revision number.

KeyboardMouse

This function is used with the MenuSelect, DragWindow, and GrowWindow commands only. Calling one of those commands immediately after calling KeyboardMouse causes the command to operate in keyboard mouse emulation mode, where the user can control the cursor motion by means of the keyboard. The KeyboardMouse function has no parameters.

```
Function KeyboardMouse;
```

Cursor Commands

These commands control the appearance of the cursor.

SetCursor

```
Procedure SetCursor ( new_ch :  integer );
```

new_ch is the character to use as cursor.

ShowCursor

```
Procedure ShowCursor;
```

HideCursor

```
Procedure HideCursor;
```

ObscureCursor

```
Procedure ObscureCursor;
```

Event Handling Commands

These commands deal with the event queue.

CheckEvents

```
Procedure CheckEvents;
```

GetEvent

```
Procedure GetEvent ( var event :  type_event );
```

event is returned with:

```
evt_kind set to the event type.
char1 set to event byte 1:  X coordinate or key value.
char2 set to event byte 2:  Y coordinate or key modifier.
```

PostEvent

```
Procedure PostEvent ( var event :  type_event );
```

event should be supplied with:

```
evt_kind set to the event type.
char1 set to event byte 1:  X coordinate or key value.
char2 set to event byte 2:  Y coordinate or key modifier.
```

FlushEvents

```
Procedure FlushEvents;
```

SetKeyEvent

```
Procedure SetKeyEvent ( chk_keyboard :  boolean );
```

chk_keyboard is the sk (set keyevent) parameter:

false= don't check keyboard
true= check the keyboard

PeekEvent

```
Procedure PeekEvent ( var event :  type_event );
```

event is returned with:

evt_kind set to the event type.
char1 set to the event byte 1: X coordinate or key value.
char2 set to the event byte 2: Y coordinate or key modifier.

Menu Commands

These commands handle menu selection and display.

InitMenu

```
Procedure InitMenu ( save_buffer, buf_size :  integer );
```

save_buffer is a pointer to the save area.
buf_size is the save area size.

SetMenu

```
Procedure SetMenu ( var my_menu_bar :  menu_bar );
```

my_menu_bar is the menu bar structure. The procedure obtains the pointer to the structure for you.

MenuSelect

```
Procedure MenuSelect ( var menu_id, menu_choice :  integer );
```

menu_id is the menu ID number.
menu_choice is the menu item number.

MenuKey

```
Procedure MenuKey ( var menu_id, menu_choice :  integer;
var key_event :  type_event );
```

menu_id is the menu ID number.
menu_choice is the item number.
key_event is returned with:

```
      char1 as the key value
      char2 as the key modifier
```

HiliteMenu

```
Procedure HiliteMenu ( menu_id :  integer );

menu_id is the menu ID number.
```

DisableMenu

```
Procedure DisableMenu ( menu_id :  integer; disable :  boolean );

menu_id is the menu ID number.
disable is the dis (disable) parameter:
      false= enable
      true= disable
```

DisableItem

```
Procedure DisableItem ( menu_id, item_num :  integer;
      disable :  boolean );

menu_id is the menu ID number.
item_num is the item number.
disable is the dis (disable) parameter:
      false= enable
      true= disable
```

CheckItem

```
Procedure CheckItem ( menu_id, item_num :  integer;
      check :  boolean );

menu_id is the menu ID number.
item_num is the item number.
check is the ck (check) parameter:
      false= turn checkmark off
      true= turn checkmark on
```

SetMark

```
Procedure SetMark ( menu_id, item_num:  integer; mark_on:  boolean;
      mark_char:  char);

menu_id is the menu ID number.
item_num is the menu item number.
```

mark_on is the mark on parameter.
mark_char is the mark char parameter.

Window Commands

These commands deal with window selection and display.

InitWindowMgr

```
Procedure InitWindowMgr ( drag_buffer, buf_size :  integer );
```

drag_buffer is the pointer to the buffer.
buf_size is the buffer size.

OpenWindow

```
Procedure OpenWindow ( var my_Winfo:  winfo );
```

my_Winfo is the Winfo data structure.

CloseWindow

```
Procedure CloseWindow ( window_id:  integer );
```

window_id is the window ID number.

CloseAll

```
Procedure CloseAll;
```

GetWinPtr

```
Procedure GetWinPtr ( window_id: integer; var winfo_ptr: integer );
```

window_id is the ID number of the window.
winfo_ptr is a pointer to the Winfo data structure.

FindWindow

```
Procedure FindWindow ( pointx, pointy:  integer; var area:  type_area;
      var window_id:  integer );
```

pointx is the X coordinate of the point.
pointy is the Y coordinate of the point.

area is the type_area (region type) parameter.
window_id is the window ID number.

FrontWindow

```
Procedure FrontWindow ( var window_id:  integer );
```

window_id is the window ID number.

SelectWindow

```
Procedure SelectWindow ( window_id:  integer );
```

window_id is the window ID number.

TrackGoAway

```
Procedure TrackGoAway ( var makeitgoaway:  boolean );
```

makeitgoaway is the go away status:
0 = not in Go-Away Box
1 = mouse was in Go-Away Box

DragWindow

```
Procedure DragWindow ( window_id, mousex, mousey:  integer );
```

window_id is the window ID number.
mousex is the mouse X coordinate.
mousey is the mouse Y coordinate.

GrowWindow

```
Procedure GrowWindow( var makeitgrow:  boolean );
```

makeitgrow is the return status:
0 = window did not grow
1 = window did grow

WindowToScreen

```
Procedure WindowToScreen ( window_id, windowx, windowy:  integer;
      var screenx, screeny:  integer );
```

window_id is the window ID number.
windowx is the window X coordinate.

windowy is the window Y coordinate.
screenx is the screen X coordinate.
screeny is the screen Y coordinate.

ScreenToWindow

```
Procedure ScreenToWindow ( window_id, screenx, screeny:  integer; var
      windowx, windowy:  integer );
```

window_id is the window ID number.
screenx is the screen X coordinate.
screeny is the screen Y coordinate.
windowx is the window X coordinate.
windowy is the window Y coordinate.

WinChar

```
Procedure WinChar ( window_id, windowx, windowy:  integer;
      my_char:  char );
```

window_id is the window ID number.
windowx is the window X coordinate.
windowy is the window Y coordinate.
my_char is the character to display.

WinString

```
Procedure WinString ( window_id, windowx, windowy:  integer;
      my_string:  string );
```

window_id is the window ID number.
windowx is the window X coordinate.
windowy is the window Y coordinate.
my_string is the string to write.

WinText

```
Procedure WinText ( window_id, windowx, windowy, text_buffer,
      textlength:  integer );
```

window_id is the window ID number.
windowx is the X coordinate in the window.
windowy is the Y coordinate in the window.
text_buffer is the pointer to the first character of text.
textlength is the number of characters to display.

WinBlock

```
Procedure WinBlock ( window_id:  integer; var my_dinfo:  dinfo;
      startx, starty, stopx, stopy:  integer );
```

window_id is the window ID number.
my_dinfo is the document information structure.
startx is the X coordinate of the upper-left corner.
starty is the Y coordinate of the upper-left corner.
stopx is the X coordinate of the lower-right corner.
stopy is the Y coordinate of the lower-right corner.

WinOp

```
Procedure WinOp ( window_id, windowx, windowy:  integer;
      opcode:  byte);
```

window_id is the window ID number.
windowx is the window X coordinate.
windowy is the window X coordinate.
opcode is the code for the operation to perform.

Control Region Commands

These commands deal with the control regions in the front window: the horizontal and vertical scroll bars, including the Thumbs.

FindControl

```
Procedure FindControl ( windowx, windowy:  integer;
      var whichctl:  ctlarea; var whichpart:  ctlpart );
```

windowx is the window X coordinate.
windowy is the window Y coordinatei.
whichctl is the control region.
whichpart is the part of the control region.

SetCtlMax

```
Procedure SetCtlMax ( whichctl:  ctlarea; newmax:  integer );
```

whichctl is the control region.
newmax is the new maximum value.

TrackThumb

```
Procedure TrackThumb ( whichctl:  ctlarea; var thumbpos:  integer;
      var thumbmoved:  boolean );
```

whichctl is the control region.
thumbpos is the Thumb position.
thumbmoved is the return status:
    0 = Thumb didn't move, thumbpos not valid
    1 = Thumb did move

UpdateThumb

```
Procedure UpdateThumb ( whichctl:  ctlarea; thumbpos:  integer );
```

whichctl is the control region.
thumbpos is the new Thumb position.

ActivateCtl

```
Procedure ActivateCtl ( whichctl:  ctlarea; makeactive:  boolean );
```

whichctl is the control region.
makeactive is the state to make the control region:
    0 = inactive
    1 = active

Utility Functions

In addition to a call for each of the Tool Kit commands, there is a utility function for obtaining the address of a Pascal variable.

PointerTo

This function obtains the address of the specified variable and returns it as the function value.

```
Function PointerTo(var Variable) :  integer;
```

Chapter 5

The Applesoft Interface

The Applesoft Interface for the MouseText Tool Kit is a set of commands that are added to the standard Applesoft commands by means of the ampersand hook. So that you can use it with other ampersand packages, the Applesoft Interface saves the existing ampersand hook address and passes any unrecognized ampersand commands on. If there is another ampersand package, that package then gets control and can test the commands. If there is no other ampersand package, then Applesoft gets control and issues a SYNTAX ERROR message.

Installing the Applesoft Interface

You'll need the relocating loader from the ProDOS Assembler Tools to load both the MouseText Tool Kit routines and the Applesoft Interface that contains the ampersand commands. The procedure for loading the Mouse Tool Kit is as follows:

(1) Load the MouseText Tool Kit from file MTXKIT.OBJ using RBOOT.

(2) Load the Applesoft Interface from file MTXAMP.OBJ using RBOOT.

(3) Write the starting address of the Tool Kit in the first two bytes of the Applesoft Interface (—not the other way around, now!).

(4) Start the Applesoft Interface by a CALL to its address plus two.

One way to do this in Applesoft looks like this:

```
10  PRINT  CHR$ (4);"BRUN RELEASE"
20 A1 = 0:A2 = 0
30  PRINT  CHR$ (4);"BRUN RBOOT"
40 A1 = USR (0),"MTXKIT.OBJ"
50 A2 = USR (0),"MTXAMP.OBJ"
60  IF A1 < 0 THEN A1 = A1 + 65536
70  IF A2 < 0 THEN A2 = A2 + 65536
80 I =  INT (A1 / 256)
90 J = A1 - I * 256
100  POKE A2,J
110  POKE A2 + 1,I
120  CALL A2 + 2
130  END
```

The MouseText Tool Kit routines are in the file named MTXKIT.OBJ; the Applesoft Tool Kit ampersand routines are in the file named MTXAMP.OBJ.

Using the Ampersand Commands

The Applesoft Tool Kit interface does not treat string variables the same as numeric variables. The interface routines copy numeric variables into internal buffers, so altering the values of those variables after an ampersand command will not change anything that the Tool Kit is doing. String variables, on the other hand, are not copied into buffers, so changing a BASIC string variable will cause changes in the display the next time the Tool Kit redisplays the menu or window with the changed string.

Note: All input parameters can be either variables or expressions, but output parameters must, of course, be variables.

The Ampersand Commands

The Applesoft Interface includes an ampersand call for each of the Tool Kit commands except for a few, such as PascIntAdr, which is used only with Pascal. There are also two utility commands, &STCNTNT (Set Content), and &DSKTPERR (Desktop Error).

The names of the ampersand commands are not the same as the command names listed in the other chapters. This is to avoid having Applesoft tokenize certain letter combinations, thereby altering a program and its listing. You must spell the command names exactly as shown or the Applesoft Interface won't recognize them and Applesoft will give you a SYNTAX ERROR message.

WARNING
A misspelled ampersand command can cause Applesoft to hang if it occurs after a RUN command. When Applesoft fails to recognize the misspelled ampersand command, it jumps back to the RUN statement, thereby getting itself into a loop condition.

By The Way: The variable names shown here are only suggestions; you may use any variable names you choose.

Startup Commands

A program normally calls these commands once to set up its operating environment.

StartDesktop

```
&STRTDSKTP(ID%,SID%,SN%,IU%,COL%)
ID% = machine ID
SID% = subsidiary ID
SN% = slot number (input and output).  If SN% = 0, StartDeskTop
      searches for a mouse card and returns the slot number in SN%.
IU% = interrupt usage:
      0 = Passive Mode
      1 = Interrupt Mode
```

COL% = number of text columns:
0 = 40 columns
1 = 80 columns

StopDeskTop

In addition to making the appropriate call to the Tool Kit, the Applesoft version of the StopDeskTop command disconnects the Applesoft Interface from the ampersand hook and restores the previous address.

Tool Kit ampersand commands will not work after a call to &STPDSKTP unless you reconnect the Tool Kit by means of the Applesoft command CALL A2+2, where A2 is the starting address of the Tool Kit Applesoft Interface. If you do stop and reconnect, you don't get back any used memory pages; for that, you have to run RELEASE and reload the Tool Kit and the Applesoft Interface.

To make sure that the ampersand hooks get properly restored when the program ends, there must be a call to StopDeskTop. Your program should include an ONERR call to &DSKTPERR, then a call to &STPDSKTP.

&STPDSKTP
(no parameters)

Version

&VRSN(V%,R%,AV%,AR%)
V% = version number
R% = revision number
AV% = Applesoft Interface version number
AR% = Applesoft Interface revision number

KeyboardMouse

The application calls this command immediately before MenuSelect, DragWindow, or GrowWindow to make those commands run in keyboard mouse emulation mode. Note that the KeyboardMouse command has no parameters.

&KYBRDMSE

Cursor Commands

These commands control the appearance of the cursor.

SetCursor

&STCRSR(CC%)
CC% = cursor character (ASCII code)

ShowCursor

&SHWCRSR
(no parameters)

HideCursor

&HDCRSR
(no parameters)

ObscureCursor

&OBCRSR
(no parameters)

Event-Handling Commands

These commands deal with the event queue.

CheckEvents

&CHCKEVNTS
(no parameters)

PostEvent

&PSTEVNT(ET%,E1%,E2%)
ET% = event type (input):
    0 = no event
    1 = button down
    2 = button up
    3 = key pressed
    4 = drag event
    5 = Apple key down
E1% = X coordinate or key value (input)
E2% = Y coordinate or key modifier (input)

GetEvent

```
&GTEVNT(ET%,E1%,E2%)
ET% = event type:
      0 = no event
      1 = button down
      2 = button up
      3 = key pressed
      4 = drag event
      5 = Apple key down
E1% = X coordinate or key value
E2% = Y coordinate or key modifier
```

FlushEvents

```
&FLSHEVNTS
(no parameters)
```

SetKeyEvent

```
&STKYEVNT(SK%)
SK% = Setkey flag:
      0 = don't check keyboard
      1 = check keyboard
```

PeekEvent

```
&PKEVNT(ET%,E1%,E2%)
ET% = event type:
      0 = no event
      1 = button down
      2 = button up
      3 = key pressed
      4 = drag event
      5 = Apple key down
E1% = x coordinate or key
E2% = y coordinate or key modifier
```

Menu Commands

These commands handle menu selection and display.

InitMenu

The InitMenu command sets aside memory space needed for the Menu Data Structure and for saving the part of the display obscured by menus.

You can determine the amount of memory space to reserve for menu displays by calculating the screen area of the largest menu in the program. The largest menu could have a large screen area because it has many items, or it could have only a few items, each of which is very long.

You calculate the screen area of a menu by taking the product of the number of items in the menu, plus one, times five bytes more than the length of the longest item string in that menu. If you are using keys to select items, each item string must include three bytes to display a space, an Apple key, and the key that selects the item. A page is 256 bytes, so to find the number of pages required, divide the size of the largest menu by 256 and round to the next highest integer.

To calculate the amount of memory to set aside for the Menu Data Structures, in bytes, add fourteen bytes for each menu plus six bytes for each item, plus two. Divide the result by 256 and round to the next highest integer to find the number of pages required. If you don't make this parameter large enough, you'll get garbage in the display when you open the menu.

```
&INITMNU(P1%,P2%)
P1% = number of pages to set aside for menu area buffer
P2% = number of pages to set aside for menu data structure
```

SetMenu

```
&STMNU(N%,M%,MI%,NA$,OB%,KC%)
N% = number of menus
M% = maximum number of items in any menu
MI% = menu information array, DIM MI%(1,N%),

where:
      MI%(0,n) = menu ID of nth menu
      MI%(1,n) = number of items in nth menu
NA$ = name array, DIM NA$(M%,N%),
where:
      NA$(0,n) = title of nth menu
      NA$(m,n) = name of mth item in nth menu
OB% = option byte array, DIM OB%(M%,N%),
where:
      OB%(0,n) = option byte of nth menu (see Table 2-2)
      OB%(m,n) = option byte of mth item in nth menu (see Table 2-4)
KC% = key character array, DIM KC%(M%,N%),
where:
      KC%(m,n) = both key characters for the mth item
      in the nth menu. Both key characters are stored in a single
      integer element of the form 256*(char1)+(char2).
```

Note: The key character array must be dimensioned even if you don't use it.

MenuSelect

```
&MNUSLCT(ID%,IN%)
ID% = menu ID number
IN% = item number
```

MenuKey

```
&MNUKY(K%,KM%,ID%,IN%)
K% =  character typed (ASCII)
KM% = key modifier
ID% = menu ID number
IN% = item number
```

By the way: The parameters are not in the same order as in the machine-language call.

HiLiteMenu

```
&HILTMNU(ID%)
ID% = menu ID number: 0 = turn off highlighting
```

DisableMenu

```
&DSABLMNU(ID%,DIS%)
ID% = menu ID number
DIS% = disable flag:
      1 = disable
      0 = enable
```

DisableItem

```
&DSABLITM(ID%,IN%,DIS%)
ID% = menu ID number
IN% = item number
DIS% = disable flag:
       1 = disable
       Ø = enable
```

CheckItem

```
&CHCKITM(ID%,IN%,CK%)
ID% = menu ID number
IN% = item number
CK% = check status:
       Ø = turn item off
       1 = turn item on
```

SetMark

```
&STMRK(ID%,IN%,MF%,MC%)
ID% = menu ID number
IN% = item number
MF% = mark flag:
       Ø = don't use mark
       1 = use mark
MC% = mark character (ASCII)
```

Window Commands

These commands deal with window selection and display.

InitWindowMgr

The InitWindowMgr command sets aside memory space needed for the Window Information Data Structure and for saving the part of the display obscured by the outline of the window.

You can determine the amount of memory space to reserve for the outline of the window by by calculating the perimeter of the largest possible window. The perimeter is the sum of twice the height plus twice the width. A page is 256 bytes, so to find the number of pages required, divide the perimeter of the largest window by 256 and round to the next highest integer.

To calculate the amount of memory to set aside for the Window Information Data Structures, in bytes, allow 42 bytes times the maximum number of windows that can be open at the same time. Divide

the result by 256 and round to the next highest integer to find the number of pages required.

```
&INITWM(P1%,P2%)
P1% = number of pages to set aside for window area buffer
P2% = number of pages to set aside for Winfo data structure
```

OpenWindow

Note: The dimensioned variable called WI% here can only be one dimensional.

```
&OPNWNDW(WI%,TS$,CS$)
WI% = window information array, DIM WI%(18),
where:
      WI%(0) is reserved
      WI%(1) = window ID number
      WI%(2) = window option byte (see Table 2-6)
      WI%(3) = window X coordinate
      WI%(4) = window Y coordinate
      WI%(5) = current content width
      WI%(6) = current content length
      WI%(7) = minimum content width
      WI%(8) = maximum content width (>0)
      WI%(9) = minimum content length
      WI%(10) = maximum content length (>0)
      WI%(11) = horizontal ctl option (see Table 2-7)
      WI%(12) = vertical ctl option (see Table 2-7)
      WI%(13) = horizontal scroll maximum
      WI%(14) = horizontal Thumb pos
      WI%(15) = vertical scroll maximum
      WI%(16) = vertical Thumb pos
      WI%(17) = contents X offset
      WI%(18) = contents Y offset
TS$ = title string
CS$ = content string array: a one-dimensional string array, where
each element is one row of the contents of the window
```

CloseWindow

```
&CLSWNDW(ID%)
ID% = ID of window to be closed
```

CloseAll

```
&CLSALL
(no parameters)
```

FindWindow

```
&FDWNDW(X%,Y%,T%,ID%)
X% = X coordinate (in mouse coordinates)
Y% = Y coordinate (in mouse coordinates)
T% = type of area point is in:
      0 = desktop
      1 = menu bar
      2 = content region
      3 = drag bar
      4 = Grow Box
      5 = Go-Away Box
ID% = window ID if in window: 0 = not in a window
```

FrontWindow

```
&FRNTWNDW(ID%)
ID% = ID number of front window
```

SelectWindow

```
&SLCTWNDW(ID%)
ID = window ID number
```

TrackGoAway

```
&TRCKGA(GF%)
GF% = GoAway function:
      0 = window should not close
      1 = window should close
```

DragWindow

```
&DRGWNDW(ID%,X%,Y%)
ID% = window ID
X% = X coordinate
Y% = Y coordinate
```

GrowWindow

Note: After you change the size of a window, you'll need to call &STCNTNT to redisplay its contents.

```
&GWNDW(ST%)
ST% = Status:
      0 = size didn't change
      1 = size changed
```

WindowToScreen

```
&WN2SCR(ID%,WX%,WY%,SX%,SY%)
ID% = window ID
WX% = window X coordinate
WY% = window Y coordinate
SX% = screen X coordinate
SY% = screen Y coordinate
```

ScreenToWindow

```
&SCR2WN(ID%,SX%,SY%,WX%,WY%)
ID% = window ID
SX% = screen X coordinate
SY% = screen Y coordinate
WX% = window X coordinate
WY% = window Y coordinate
```

SetContent

This ampersand command changes the contents and content offsets in a window definition. It is typically used to redisplay the contents of a window after a call to GrowWindow. It can also be used for scrolling the contents of a window by making the content string the same as before and changing the X and Y offset.

You should not expect the window position with coordinates X%,Y% to contain the X%th character in the Y%th element of the content string array. Remember that there are offsets OX% and OY%, and that the window Y coordinate of the first column in a window has the value Ø, not 1. You will normally have to perform some arithmetic to figure out which character in the content string array corresponds to a given window position.

Note: The SetContent command subsumes the functions of WinString, WinChar, WinText, and WinBlock. Of those commands, the Applesoft Interface includes as separate commands only WinString and WinChar.

&STCNTNT(ID%,RE%,CS$,OX%,OY%)
ID% = window ID number
RE% = a reserved variable (should be set = Ø)
CS$ = new content string (see &OPNWNDW command)
OX% = new X offset into content (replaces value set by WI%(17))
OY% = new Y offset into content (replaces value set by WI%(18))

WinChar

Please refer to the notes under the WinOp command.
&WNCHR(ID%,X%,Y%,CH%)
ID% = window ID
X% = X coordinate of position in window
Y% = Y coordinate of position in window
CH% = ASCII code for character

WinString

Please refer to the notes under the WinOp command.
&WNSTR(ID%,X%,Y%,S$)
ID% = window ID
X% = X coordinate of position in window
Y% = Y coordinate of position in window
S$ = character string

Parameter Note: While all the WinString parameters are inputs, SS cannot be an expression, although it can be either a simple variable or an array element.

WinOp

```
&WNOP(ID%,X%,Y%,OC%)
ID% = window ID
X%  = X coordinate of position in window
Y%  = Y coordinate of position in window
OC% = operation code:
      26 = clear from start of window to (but not including)
           position X,Y.
      27 = clear from start of line to (but not including)
           position X,Y
      28 = clear entire window
      29 = clear from position X,Y to end of window
      30 = clear line
      31 = clear from position X,Y to end of line
```

Important Note: The window commands &WNCHR, &WNSTR, and &WNOP change the contents of the window in the display, but they do not change the content string array (called CSS in this manual) that is used to update the window after it has been moved, resized, re-exposed, or the like. It is up to the application to update the content string array to match the new content of the window. (It is not necessary to call &STCNTNT under these circumstances.)

You should not expect the window position with coordinates X%,Y% to contain the X%th character in the Y%th element of the content string array. Remember that there are offsets OX% and OY%, and that the window Y coordinate of the first column in a window has the value 0, not 1. You will normally have to perform some arithmetic to figure out which character in the content string array corresponds to a given window position.

Control Region Commands

These commands deal with the control regions in the front window: the horizontal and vertical scroll bars, including the thumbs.

ActivateControl

```
&ACTVTCTL(CTL%,DIS%)
CTL% = which control region
       1 = vertical scroll bar
       2 = horizontal scroll bar
DIS% = disable flag
       0 = disable
       1 = enable
```

FindControl

```
&FDCTL(WX%,WY%,CTL%,PC%)
WX% = window X coordinate
WY% = window Y coordinate
CTL% = control region point is in:
       0 = content
       1 = vertical scroll bar
       2 = horizontal scroll bar
       3 = none of the above
PC% = part of the control point is in:
       0 = inactive control
       1 = Up/Left-Arrow
       2 = Down/Right-Arrow
       3 = page up/left region
       4 = page down/right region
       5 = Thumb
```

SetCtlMax

```
&STCTLMX(CTL%,NM%)
CTL% = control to set new maximum for:
       1 = vertical scroll bar
       2 = horizontal scroll bar
NM% = new maximum for control range (must be > 1)
```

TrackThumb

```
&TRCKTHMB(CTL%,TP%,MF%)
CTL% = which control to update Thumb for:
       1 = vertical scroll bar
       2 = horizontal scroll bar
TP% = new Thumb position
```

```
MF% = move flag:
      0 = Thumb didn't move
      1 = Thumb did move
```

UpdateThumb

```
&UPDTTHMB(CTL%,TP%)
CTL% = which control to update Thumb for:
       1 = vertical scroll bar
       2 = horizontal scroll bar
TP% = new Thumb position
```

Utility Commands

These commands provide utility functions not included in the standard Tool Kit commands.

Get Window Info

This ampersand command fills the WI% array with the current values of the array originally set in &OPNWNDW. An application program can use it to obtain values changed by the Tool Kit by user actions—for example, current width and length after a call to &GWNDW (GrowWindow). The WI% array need not be the same as the one in the original call to &OPNWNDW.

Important: The dimensioned variable called WI% here must be dimensioned to at least eighteen; otherwise, the call will return an error.

```
&GTWNFO(ID%,WI%)
ID% = window ID number
WI% = window information array, as dimensioned in &OPNWNDW.
```

DesktopError

Errors that are generated by the Tool Kit return an Applesoft error number 53, ILLEGAL QUANTITY; when this happens, a call to &DSKTPERR will return the Tool Kit command and error number. The Applesoft Interface itself can also generate other Applesoft errors such as SYNTAX ERROR and OUT OF MEMORY.

```
&DSKTPERR(CN%,EN%)
CN% = command number of the last call made to the Tool Kit
EN% = error code returned by that command: Ø = no errors
```

NextWindow

Starting with &FRNTWNDOW and using this call repeatedly until I2% = Ø, the program can select each window on the screen, in order of depth, testing or changing them as it goes.

```
&NXTWNDW(I1%,I2%)
I1% = input window ID number.  I1% = Ø selects front window.
I25 = output ID of the next window.  If none, I2% = Ø.
```

Set Interrupt Mask

This command sets the interrupt mask bit in the 65Ø2's status byte. If IB% = 1, the bit is set—that is, interrrupts are disabled. If IB% = Ø, the bit is cleared and interrupts are enabled.

WARNING
This is a dangerous command. It does not save or restore anything, nor does it update any status information in the Tool Kit. It is included for debugging, and may not be in the final version of the Tool Kit.

```
&STIMB(IB%)
IB% = value to set interrupt mask bit (Ø or 1)
```

Appendix A

The AppleMouse II Interface Card

To use the Apple mouse with an Apple II, Apple II Plus, or Apple IIe, you need the AppleMouse II Interface Card installed in one of the expansion slots (Apple recommends using slot 4). Like most Apple peripheral cards, it contains I/O firmware that is executed by the 6502 central processor whenever you access the slot. The mouse interface card also contains its own microprocessor with firmware and a timer. The microprocessor on the card keeps track of the position of the mouse and the state of the button on the mouse. The microprocessor handles the transfer of mouse information and other communications between the card and the central processor.

Passive Versus Active Operation

Most positioning devices used with the Apple II, such as the joystick and the graphics tablet, are passive devices: they don't require any processing until an application program requests information from them. The mouse, on the other hand, is an active device, at least at the hardware level: movement of the mouse requires immediate attention to keep the system from losing track of its position and direction.

A computer normally handles this need for immediate response by means of interrupts. When the mouse is moved rapidly, it generates interrupts often enough to have a significant impact on the computer's operation. If the computer is engaged in other tasks that are dependent on precise timing, as the Apple II often is, the added burden of processing the interrupts from the mouse can be intolerable.

To reduce the interrupt burden on the Apple II's processor, the AppleMouse II uses an intelligent interface card. The card has an MC6805 microprocessor that is dedicated to keeping track of the mouse, thus making it possible for the AppleMouse II to operate as either an active device or a passive device. In the Passive Mode, the MC6805 determines the instantaneous movement and direction of the mouse and stores the information on the card until the processor in the Apple II requests the information. Thus, the AppleMouse II can act like a

passive device in applications that cannot tolerate interrupts, or, for applications where interrupts are appropriate, it can operate as an active device.

Mouse Interrupts

One reason to use the mouse in Interrupt Mode is to be able to move a cursor on the display screen without the flicker produced by updating the cursor during the wrong part of the display refresh cycle. In Interrupt Mode, the AppleMouse II generates interrupts that are synchronized with the vertical blanking interval.

The Apple IIe has a signal named VBL, but it isn't available as an interrupt. The VBL signal is not available at all on an Apple II or Apple II Plus, so the mouse card has a hardware timer that it uses to generate interrupts synchronized with the vertical blanking interval.

Because the AppleMouse II transmits an interrupt request only at the beginning of a vertical blanking interval, it cannot generate interrupts faster than 60 times per second. This limits the number of mouse interrupts and keeps the mouse from monopolizing the central processor.

The TimeData Firmware Call

There is a little-used call in the firmware on the AppleMouse II card. That call sets the interrupt rate to either 50 or 60 Hz. The default is 60 Hz., which keeps the VBL interrupts the card generates in step with the true VBL on a North American Apple II. For European machines, the VBL rate is 50 Hz.

The low byte of the TimeData entry-point address is $Cn1C. Input data is in the accumulator. With the accumulator set to 0, TimeData sets the VBL rate to 60 Hz. With the accumulator set to 1, the call sets the VBL rate to 50 Hz. The only valid accumulator contents for this call are 0 and 1. On output, the carry bit is clear and the screen holes are unchanged.

You should call TimeData just before calling InitMouse. If you do not call TimeData first, the VBL rate will be set to 60 Hz when you call InitMouse.

Appendix B

The Mouse Firmware Interface

On the Apple IIc, the interface hardware and firmware for the AppleMouse II is built in. On the Apple IIe, the user must install a mouse interface card in order to use the AppleMouse II. The interface card for the AppleMouse II contains the firmware that communicates with and controls the mouse hardware.

The Apple II MouseText Tool Kit uses the mouse firmware in the Apple IIc or in the card in the Apple IIe to operate the mouse. This appendix describes the interface to the firmware.

Note: If you do all your mouse operations via Tool Kit commands, you do not need to communicate directly with the mouse firmware and so do not need to learn the material in this appendix.

Finding the Mouse Card

The AppleMouse II interface card can be installed in any peripheral slot except slot 0; use of slot 4 is recommended but not required. The firmware on the card stores signature bytes in five of the memory locations assigned to the slot it is in. The addresses and values of the signature bytes are as follows:

Address	Value
$CnØ5	$38
$CnØ7	$18
$CnØB	$Ø1
$CnØC	$2Ø
$CnFB	$D6

The letter n in the addresses stands for the slot number. Your program can determine which slot the mouse card is in by reading the memory locations for each value of n from 1 to 7 and comparing the values with the values shown above.

Reading Mouse Data

The mouse firmware stores position and status information in the display buffer locations reserved for the slot the mouse card is in (the screen holes, also called mouse holes). When you call the ReadMouse routine or the ServeMouse routine (described later in this appendix), the firmware updates the information in the mouse holes. Your program can address these locations by using the slot number as an index, as indicated by the letter n in Table B-1.

By the way: Chapter 6 of the Apple IIe Reference Manual describes the way you address the reserved screen locations.

WARNING
If your program ever uses the auxiliary memory in the Apple IIe, be sure that you get all the switches set back to main memory before you use the Tool Kit. If you write data into the reserved screen locations in the auxiliary memory, not only will the mouse firmware not read them, but you may cause other firmware to malfunction (spelled c-r-a-s-h).

Table B-1. Screen Locations for Mouse Data

Address	Contents
$478 + n	Low byte of X position
$4F8 + n	Low byte of Y position
$578 + n	High byte of X position
$5F8 + n	High byte of Y position
$678 + n	(used by the firmware)
$6F8 + n	(used by the firmware)
$778 + n	Button and interrupt status
$7F8 + n	Current Operating Mode

In its normal operating position (oriented with its cable directed away from the user), the value of the X position coordinate increases as the mouse is moved to the right and the value of the Y position coordinate increases as the mouse is moved toward the user. The maximum values of X and Y are -32768 to +32767, but the firmware normally clamps them to the range 0 to +1023 ($0 to $3FF). You can change the clamping range by calling the ClampMouse routine, which is described later in this appendix.

The smallest mouse movement that the mouse hardware can detect is one count in either the X or Y direction; that is equivalent to about 0.01 inch (0.3 mm). The largest movement that the hardware can handle is 16 bits in either axis. A change of position from -32768 to +32767 corresponds to about 60 feet of mouse movement.

The bits in the button and interrupt status byte are assigned as shown in Table B-2, where a value of 1 means the function is true.

Table B-2. Button and Interrupt Status Byte

Bit #	Function
7	Button is down
6	Button was down at last reading
5	Mouse moved since last reading
4	(used by the firmware)
3	Video blanking interrupt
2	Button press interrupt
1	Mouse movement interrupt
Ø	(used by the firmware)

Operating Modes

When you turn on the power, the firmware comes up in the off condition with its X and Y position registers set to Ø. You activate the firmware by loading the accumulator with a mode byte and calling the SetMouse routine. The settings of the bits in the mode byte determine the mode of operation, as shown in Table B-3.

Table B-3. Bits in the Mode Byte

Bit #	Function
7-4	(used by the firmware)
3	Enable interrupt on video blanking (VBL)
2	Enable interrupt on next VBL after button pressed
1	Enable interrupt on next VBL after mouse movement
Ø	Turn on the mouse

You can enable any combination of interrupts by setting the appropriate bits in the mode byte. You can set mode combinations that don't make sense, such as $Ø2: Mouse Off plus Enable Interrupt On Mouse Movement, which acts just like $ØØ: Mouse Off.

Setting the low bit in the mode byte to Ø turns off certain functions of the mouse: the mouse position is not tracked, calls to ReadMouse don't update the status byte or the screen holes, and button and movement interrupts are not generated. Other mouse functions will work as usual: PosMouse and ClearMouse will change the mouse position data, ClampMouse will set new values, and so on. Turning the mouse on and off by changing the mode byte does not reset any mouse values.

WARNING
You must not set the high bits of the mode byte. Mode byte values greater than $ØF will cause the SetMode routine to return an illegal-mode error.

Passive Mode

Calling the SetMouse routine with a mode byte of $Ø1 puts the firmware into Passive Mode (no interrupts occur). Passive mode is the simplest way to use the mouse, and it is the only way to use it in systems with peripherals that cannot tolerate interrupts.

In Passive Mode, the interface card stores mouse information without affecting the operation of the CPU. When your program calls the ReadMouse routine, the firmware updates the mouse information in the screen locations, where your program can read it.

Interrupt Mode

If your program uses interrupts it must include an interrupt handling routine that calls the ServeMouse routine. The ServeMouse routine determines whether the interrupt was caused by the mouse. If it was, the ServeMouse routine calls ReadMouse.

Depending on the setting of the mode byte, the firmware can interrupt the CPU on one or more of the following events:

- Mouse motion
- Mouse button pressed
- Display video blanking

You can set the mode byte to $08—mouse off, VBL interrupt on—to generate interrupts on display video blanking (VBL) only. Regardless of the kind of event that causes the interrupt, the mouse hardware will interrupt the CPU only at the beginning of the video blanking interval, which occurs every 60th of a second. This enables your program to update the display between screen refresh cycles and avoid making the display flicker.

Unclaimed Interrupts

There is a bug in the AppleMouse II firmware that can effect the way ServeMouse works. If the application program takes more than one video blanking cycle (normally about 16 milliseconds) to respond to a mouse-generated interrupt, there is a chance that ServeMouse will not claim the interrupt. In a ProDOS or Pascal environment, this can be fatal. There are several possible ways to avoid this problem.

One approach, if you are not working under a system like ProDOS or Pascal, is to make sure that unclaimed interrupts aren't fatal to your system and just ignore them. Another solution is to make sure that you always service interrupts within one VBL cycle (one sixtieth of a second). If you have to turn off interrupts for that long or longer, you should first use SetMouse to set the mode to 0 and call ServeMouse to clear any existing interrupt.

If you are working under an established operating system, like ProDOS or Pascal, for which unclaimed interrupts are fatal, you can use one of the following suggestions to make sure that all interrupts are claimed.

If the mouse is the only interrupting device, write your interrupt handler so that it claims all interrupts.

If the mouse is not the only interrupting device, there are three ways of handling the problem. One is to write the mouse interrupt handler to claim all unclaimed interrupts and make sure that it is installed last. Another method is to write a spurious interrupt handler (sometimes called a demon), not associated with any device, that claims all unclaimed interrupts. This interrupt handler must be installed last. The third method is to include code in every interrupt handler to determine whether that interrupt handler is last. If it is, then that interrupt handler claims any unclaimed interrupts, even if not generated by its device.

Making Calls to Mouse Firmware

Your programs make calls to the mouse firmware by means of a table that conforms to Apple Firmware Protocol 1.1, described in the Apple IIe Design Guidelines as Pascal 1.1 Protocol. Table B-4 contains the low byte of the entry address of each of the firmware routines. (The high byte of each address is $Cn, where n is the number of the slot the mouse interface card is in.) The address bytes are stored in locations $Cn12 through $Cn19, arranged as shown in Table B-4.

Table B-4. Entry Point Address Bytes

Location	Contents
$Cn12	Low byte of SetMouse entry-point address
$Cn13	Low byte of ServeMouse entry-point address
$Cn14	Low byte of ReadMouse entry-point address
$Cn15	Low byte of ClearMouse entry-point address
$Cn16	Low byte of PosMouse entry-point address
$Cn17	Low byte of ClampMouse entry-point address
$Cn18	Low byte of HomeMouse entry-point address
$Cn19	Low byte of InitMouse entry-point address

Thus, for a mouse card installed in slot 4, you can calculate the entry address for the SetMouse routine by adding $C400 to the contents of location $C412. Your program can use the values in the table to construct a jump table to use for calling the routines.

By the Way: You must disable interrupts before calling the mouse firmware.

Parameter Passing

Before calling any of the firmware routines, your program must load the X and Y index registers with the number of the slot the mouse card is in, as follows:

X index register: $Cn

Y index register: $n0

Your program passes information to certain firmware routines via the accumulator and the screen locations, as noted in the descriptions of the routines.

When your program regains control, the contents of the accumulator and the index registers will be undefined, except as noted in the descriptions of the routines. The carry bit indicates the error status of the routine just ended:

Successful execution: C = Ø

Unsuccessful execution: C = 1

The Firmware Routines

This section describes the functions of the firmware routines whose entry-point addresses are given in the previous section.

SetMouse

SetMouse starts the mouse operating in the mode indicated by the contents of the accumulator, as defined in the "Operating Modes" section earlier in this appendix. If the mode byte is greater than $ØF, the routine will return with the carry bit set to one, indicating an error. This routine does not clear the screen locations used for storing mouse data.

ServeMouse

If the pending interrupt was caused by the mouse, ServeMouse sets the status byte at location $778 + n to show what event caused the interrupt. Upon return from this routine, the carry bit is set to Ø if the interrupt was caused by the mouse; otherwise, the carry bit is set to 1. This routine does not update the other mouse screen locations.

Note: This routine is an interrupt service routine; it does not require particular values in the accumulator or the index registers.

ReadMouse

Readmouse transfers the current values of the mouse X and Y position and button data into the appropriate screen locations and sets bits 1, 2, and 3 of the status byte at location $778 + n to Ø. On return, the carry bit is Ø.

ClearMouse

ClearMouse sets the mouse's X and Y position values to zero, both on the interface card and in the screen locations. It does not change the contents of the interrupt and button status byte. On return, the carry bit is Ø.

PosMouse

PosMouse sets the mouse X and Y position to the values in the screen locations. On return, the carry bit is Ø.

WARNING
Do not change the contents of any screen locations other than the X and Y position locations.

ClampMouse

ClampMouse sets the clamping bounds for either the X or Y position value. To clamp the X direction, load the accumulator with a Ø; to clamp the Y direction, load the accumulator with a 1. Store the new bounds in the slot Ø screen locations, as follows:

$478	low byte of lower clamping bound
$4F8	low byte of upper clamping bound
$578	high byte of lower clamping bound
$5F8	high byte of upper clamping bound

On return, the carry bit is Ø and the X and Y position screen locations are undefined. To get valid position data, you have to call the ReadMouse routine.

HomeMouse

HomeMouse sets the internal position values to the upper-left corner of the clamping window. On return, the carry bit is Ø and the X and Y screen locations are changed.

InitMouse

InitMouse sets internal mouse data to default values and synchronizes the interrupt timer on the card with the display vertical blanking. On return, the carry bit is zero and the screen locations are unchanged. To get valid position data, you have to call the ReadMouse routine.

WARNING
On the Apple II plus, the InitMouse routine clears the Hi-Res screen in order to synchronize its timer with the vertical blanking, so you should display Hi-Res graphics only after you have called InitMouse.

Appendix C

The Mouse Pascal Attach Driver

What's-It-All-About Department: The material in this appendix is not part of the MouseText Tool Kit. It is included here because it is new and is not described in any existing manuals.

Installing the Mouse Pascal Attach Driver

The Pascal disk that came with the Tool Kit contains two versions of the Pascal mouse I/O attach driver. It also contains the file SYSTEM.ATTACH, which performs the attach operation each time the user starts with it on the system disk. If the mouse driver is the only one you need to attach, all you have to do is copy the appropriate files onto your system disk. Table C-1 contains a list of the files on the Pascal disk that came with the Tool Kit.

Table C-1. Attach Files

File Name	Contents of File
SYSTEM.ATTACH	The system code file that performs the attach operation each time the system is initialized.
ATTACH.DRIVERS	Driver with its own interrupt manager.
ATTACH.DATA	Data for driver with own interrupt manager.
M.ATTACH.DRIVER	Add to existing drivers with interrupts.
M.ATTACH.DATA	Data to add to drivers with interrupts.

There are two versions of the ATTACH.DATA and ATTACH.DRIVERS files, one with interrupts and one without. If the mouse is the only source of interrupts in your system, use the files named ATTACH.DRIVERS and ATTACH.DATA . If you already have other attach drivers that include an interrupt handler, you can add just the mouse driver by using the files named M.ATTACH.DRIVER and M.ATTACH.DATA . To do this, you'll have to use the Library Program to make a new ATTACH.DRIVERS file with the mouse driver added to your other attach drivers. You'll also have to execute the ATTACHUD.CODE utility program to make a new ATTACH.DATA file. For a complete description of Pascal attach drivers and the procedures to follow in installing them, see Pascal Tech Note #11.

About Pascal Attach Drivers

Pascal 1.1 and Pascal 1.2 for the Apple II include a method for adding custom I/O drivers to the system. To add a driver using this method, you have to use the programs ATTACHUD.CODE and SYSTEM.ATTACH provided by Apple.

When the system is initialized, part of the program SYSTEM.PASCAL looks for the program SYSTEM.ATTACH on the main system disk. If program SYSTEM.ATTACH is present, the system executes it before executing SYSTEM.STARTUP. SYSTEM.ATTACH, in turn, uses files named ATTACH.DATA and ATTACH.DRIVERS, which must also be on the main system disk. ATTACH.DATA is the file you created using the ATTACHUD program, and ATTACH.DRIVERS is a library file that contains all of the drivers being attached.

SYSTEM.ATTACH installs the attach drivers in the Pascal heap space below

the point where ordinary programs access it. This reduces the stack and heap space available to the program by an amount equal to the size of the drivers.

The Pascal Interface

Table C-2 shows the Pascal I/O calls for each of the mouse firmware entry points. An outline of the functions of the direct I/O calls follows.

Table C-2. Pascal I/O Calls

Firmware entry point	Direct I/O call
PINIT	none
PREAD	none
PWRITE	none
PSTATUS	none
SETMOUSE	UNITSTATUS control code Ø
SERVEMOUSE	interrupt handler
READMOUSE	UNITREAD
CLEARMOUSE	UNITCLEAR
POSMOUSE	UNITSTATUS control code 1
CLAMPMOUSE	UNITSTATUS control code 2
HOMEMOUSE	UNITSTATUS control code 3
INITMOUSE	UNITCLEAR (first time only)

UNITCLEAR
reset mouse position to Ø, Ø.
reset user interrupt address to No-op.
reset clamping to default values [(Ø, 1Ø23), (Ø, 1Ø23)].

UNITREAD
read x, y button status
The read buffer should be defined as follows:

```
ReadBuffer: record
               X:  integer;
               Y:  integer;
               Button:  integer;
            end;
```

UNITWRITE
No-op.

UNITSTATUS
CONTROL CODE 0: set Mouse Mode and user interrupt address
The control data buffer should be defined as follows:

```
Buffer: record
          MouseMode: integer;
          IntAddr: integer;
        end;
```

IntAddr should be obtained by a call to the user's interrupt handler. (See the GetIntAddr procedure in MouseInt.Text). If IntAddr is zero, the user interrupt address will be set to a No-op (RTS instruction).

CONTROL CODE 1: set mouse position
The control data buffer should be defined as follows:

```
Buffer: record
          X: integer;
          Y: integer;
        end;
```

CONTROL CODE 2: clamping
The control data buffer should be defined as follows:

```
Buffer: record
          MouseLeft: integer;
          MouseRight: integer;
          MouseTop: integer;
          MouseBottom: integer;
        end;
```

CONTROL CODE 3: Home mouse.

STATUS CODE 0: return Mouse Mode and interrupt address
The status data buffer should be defined as follows:

```
Buffer: record
          MouseMode: integer;
          IntAddr: integer;
        end;
```

STATUS CODE 1: No-op

STATUS CODE 2: return clamping values
The status data buffer should be defined as follows:

```
Buffer: record
           MouseLeft: integer;
           MouseRight: integer;
           MouseTop: integer;
           MouseBottom: integer;
        end;
```

STATUS CODE 3: No-op.

Interrupts

Call SERVEMOUSE.
Call user interrupt handler. If no user interrupt, the call defaults to a No-op (RTS instruction).

Appendix D

Sample Program

This appendix contains a sample program showing how to use the mouse and the Tool Kit. The disks that contain the Tool Kit routines also contain three versions of a sample program, in Pascal, Applesoft, and assembly language. All three sample programs are functonally the same. The pseudocode listing that follows is similar to those sample programs, but it is not identical. To find out exactly what the sample programs look like, you should list them from the disk.

The pseudocode program is an example of the way the Tool Kit is intended to be used. The program includes the following functions.

- start the desktop
- set up menus
- set up a cursor
- track the mouse
- display a pull-down menu
- enable and disable an item in a menu
- open a window
- select a window
- drag a window
- grow a window
- scroll the contents of a window
- close a window

The user stops this program by selecting the "Quit" item in the menu.

Pseudocode Listing

Here is the pseudocode listing of the program.

```
call StartDeskTop                    ; start up the Tool Kit
call InitMenu                        ; allocate screen save space
call SetMenu(DemoMenu)               ; set up our menus
call ShowCursor                      ; turn on cursor
```

```
call InitWindowMgr                    ; allocate screen save space for window
quitflag := false                     ; used to terminate program

while not quit flag do                ; main loop
; call CheckEvents                    * no longer needed in version 2 *
  call GetEvent                       ; get the next event in event queue
  case eventtype of                   ; base action on type of event returned
    button_up, no_event, drag_event, open_apple_drag_event :
      do nothing                      ; we are ignoring these
    keypress : call HandleKeys        ; handle keyboard input from user
    button_down : call HandleButton   ; handle button down on mouse
  end case
end while                             ; end of main loop
do any clean up
end program                           ; end of program

HandleKeys :                          ; character input is enter here
  if open_apple_key down do           ; check for commands
    call MenuKey                      ; translate into menu command
    call MenuCase                     ; and execute it
  end if
  return

HandleButton
  call FindWindow                     ; where did button go down ?
  case event_location of              ; base action on where it occurs
    in_desktop : do nothing
    in_menu : call HandleMenu         ; menu bar, menu operation
    in_content : call DoContent       ; content region, find out more
    in_drag_bar : call DragIt         ; drag bar, drag the window
    in_grow : call DoGrow             ; growth region, grow the window
    in_close : call CloseIt           ; close the top window
  end case
  return

HandleMenu
  call MenuSelect                     ; have toolkit perform selection
  call MenuCase                       ; execute selection
  return

MenuCase                              ; execute the menu selection
  if menu_id = 0
    then do nothing                   ; nothing selected
    else do
      case menu_id & menu_item
        do corresponding operation
      end case
      call HiliMenu(0)                ; task is done, turn off highlight
  return

DoContent                             ; button down inside a window
  call FrontWindow                    ; find front window id
```

```
  if button_down does not occurs in front window
    then call SelectWindow          ; bring that window to front
    else do
      call ScreenToWindow           ; use local coordinate
      call FindControl              ; find if it occur in control
      case point_is_in
        in_content : depend on application, nothing here
        in_vertical_scroll_bar, in_horz_scroll_bar :
          call ScrollBar            ; perform scrolling
        in_dead_zone : do nohing
      end case
  return

ScrollBar
  case where_in_scroll_bar
    arrow, page :
      scroll 1 or n lines
      call UpdateThumb              ; udpate thumb position
    thumb :
      call TrackThumb               ; let toolkit track thumb movement
      if thumb_moved then scroll accordingly
  end case
  return

DragIt
  call SelectWindow                 ; bring window to front if it is in back
  call DragWindow                   ; let toolkit follow the drag
  return

DoGrow
  call GrowWindow                   ; let toolkit follow the growth
  if size_changed do                ; if size of windwo changed extra work
    call SetCtlMax                  ; thumb position etc may be changed
    call ActivateCtl                ; scroll bar may become active/inactive
    call WinBlock                   ; window is blank afterwards, update it
  return
```

Appendix E

MouseText Characters

The character generator ROM in the Apple IIc includes a set of text icons in the alternate character set. Apple is planning to make these icon characters available on the Apple IIe as well. The primary purpose of the new icon characters is for producing interactive displays using the Apple Mouse or other pointing devices.

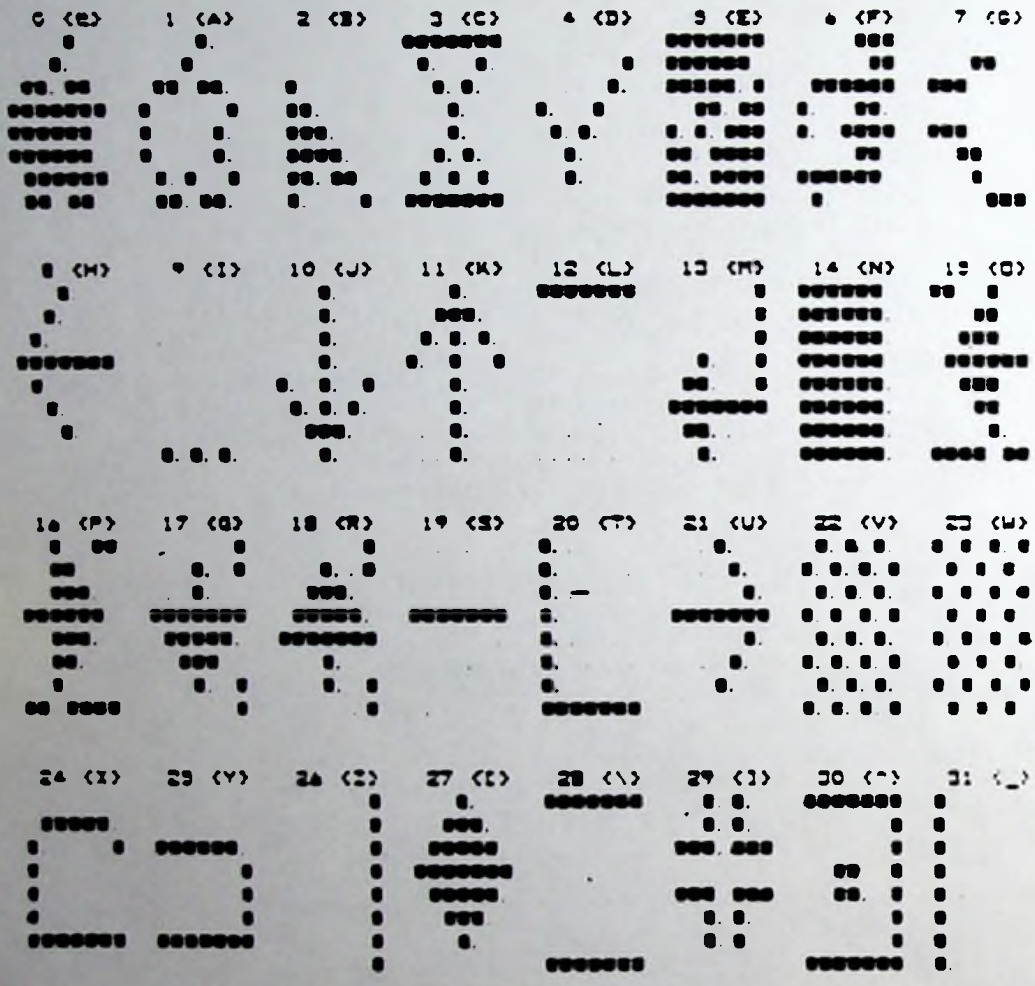
The new icon characters replace one of the sets of inverse uppercase letters (and a few special characters) in the alternate character set (selected by the ALTCHARSET soft switch). To print the icon characters with the MouseText Tool Kit installed, use the ASCII character values from 128 through 159 ($80 through $9F), as shown in Figure E-1.

ASCII Note: The Tool Kit interprets ASCII codes as follows:

0-31:	control characters or mousetext
32-127:	normal video
128-159:	MouseText characters
160-255:	inverse video

In the future, the range from 0-31 may be used as control codes only. Therefore, you should use the range from 128-159 for MouseText.

Figure E-1 The Mouse Text Icon Characters



Appendix F

Tool Kit Error Codes

Table F-1 is a cumulative list of the error codes returned in the 65Ø2's accumulator when a MouseText Tool Kit command encounters an error condition. The error codes returned by each command are listed with the commands in Chapter 2.

In addition to the error codes returned by individual commands, the first three listed here are generic error codes that can be returned by any command.

Table F-1. MouseText Tool Kit Error Codes

1 ($Ø1)	Illegal command number
2 ($Ø2)	Wrong number of parameters
3 ($Ø3)	StartDeskTop hasn't been called
4 ($Ø4)	Machine or operating system not supported
5 ($Ø5)	Invalid slot number (less than Ø or greater than 7)
6 ($Ø6)	Mouse Interface Card not found
7 ($Ø7)	Interrupt mode in use (Program specified interrupt mode in StartDeskTop, so it can't call CheckEvents.)
8 ($Ø8)	Menu ID was not found
9 ($Ø9)	Item Number is not valid
1Ø ($ØA)	Save area (from InitMenu) is too small
11 ($ØB)	Tool Kit could not install interrupt handler
12 ($ØC)	Window with same ID already open
13 ($ØD)	InitWindowMgr buffer too small for this window
14 ($ØE)	Bad Winfo -- tried to open window with ID = Ø, or conflicting max and min width or length
15 ($ØF)	Window ID number not found
16 ($1Ø)	There are no windows
17 ($11)	Error returned by user hook routine
18 ($12)	Bad control ID (not 1 or 2)
19 ($13)	Event queue full, event not posted
2Ø ($14)	Illegal event, event not posted
21 ($15)	Illegal UserHook ID number (not Ø or 1)
22 ($16)	Operation cannot be performed

Resource Central

[redacted]

P.O. Box 11250
Overland Park, Kansas 66207
913-469-6502
913-469-6507 (fax)

Resource Central, not Apple Computer,
is responsible for supporting this product.